UCRL-ID-136921

Parametric Grid Information in the DOE Knowledge Base: Data Preparation, Storage, and Access

J.R. Hipp, C.J. Young, S.G. Moore, E.R. Shepherd, C.A. Schultz and S.C. Myers

October 1, 1999

U.S. Department of Energy

Lawrence
Livermore
National
Laboratory

DISCLAIMER

Work performed under the auspices of the U. S. Department of Energy by the University of California Lawrence Livermore National Laboratory under Contract W-7405-Eng-48.

SAND99-2277
Unlimited Release
Printed October 1999

Parametric Grid Information in the DOE Knowledge Base: Data Preparation, Storage, and Access

James R. Hipp
Decision Support Systems Department

Chris J. Young
Geophysical Technology Department

Susan G. Moore
Decision Support Systems Department

Ellen R. Shepherd
Decision Support Systems Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1138

Craig A. Schultz

Stephen C. Myers

Lawrence Livermore National Laboratories
P.O. Box 808
Livermore, CA 94550

Abstract

The parametric grid capability of the Knowledge Base provides an efficient, robust way to store and access interpolatable information which is needed to monitor the Comprehensive Nuclear Test Ban Treaty. To meet both the accuracy and performance requirements of operational monitoring systems, we use a new approach which combines the error estimation of kriging with the speed and robustness of Natural Neighbor Interpolation (NNI). The method involves three basic steps: *data preparation (DP)*, *data storage (DS)*, and *data access (DA)*. The goal of data preparation is to process a set of raw data points to produce a sufficient basis for accurate NNI of value and error estimates in the Data Access step. This basis includes a set of nodes and their connectedness, collectively known as a tessellation, and the corresponding values and errors that map to each node, which we call surfaces. In many cases, the raw data point distribution is not sufficiently dense to guarantee accurate error estimates from the NNI, so the original data set must be densified using a newly developed interpolation technique known as *Modified Bayesian Kriging*. Once appropriate kriging parameters have been determined by variogram analysis, the optimum basis for NNI is determined in a process we call *mesh refinement*, which involves iterative kriging, new node insertion, and Delauny triangle smoothing. The process terminates when an NNI basis has been calculated which will fit the kriged values within a specified tolerance. In the data storage step, the tessellations and surfaces are stored in the Knowledge Base, currently in a binary flatfile format but perhaps in the future in a spatially-indexed database. Finally, in the data access step, a client application makes a request for an interpolated value, which triggers a data fetch from the Knowledge Base through the libKBI interface, a walking triangle search for the containing triangle, and finally the NNI interpolation.

We demonstrate the entire process using a synthetic data set of seismic travel times for an evenly spaced grid of 121 events which require source-to-receiver-specific path corrections to be accurately located. To create the test set, predicted IASP91 arrival times for three International Data Center primary network stations were perturbed using values taken from separate, randomly generated, order 50 spherical harmonic surfaces. 30 ground truth points were then chosen randomly within the grid area to use as ground truth data points to develop travel time corrections. After data preparation and data storage, the resultant parametric grid for the travel time corrections was accessed by the event location code EvLoc. For the seven test events which were co-located with ground truth events, mislocations were reduced from an average of 42.7 km to 2.1 km, and the semimajor axes of the error ellipses were reduced from 141.9 km to 50.9 km. Overall, the average event mislocation reductions were much smaller -- from 58.8 km to 47.0 km -- reflecting the limits imposed on the interpolation by the relatively small number of ground truth events. This dependence of interpolated value accuracy and error estimate size on proximity to ground truth points is a typical result, and reflects the underlying kriging process which bases the interpolation of correction information on the spatial correlation characteristics derived from the data itself. This is a marked improvement over surface fitting approaches which suffer either from overfitting or underfitting and provide poor if any error estimation for interpolated values.

Acknowledgments

Helpful reviews of this report were provided by Randy Simons and Eric Chael. This work was supported by the United States Department of Energy under Contract DE-AC04-94AL85000. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy.

(This page intentionally left blank)

Contents

Figures

1 Introduction

To improve the U.S. National Data Center (USNDC) capability to monitor a Comprehensive Test Ban Treaty (CTBT), researchers at the U.S. Department of Energy (DOE) national laboratories are collaborating to develop a collection of data and associated access tools collectively known as the Knowledge Base. The Knowledge Base (hereafter KBase) will consist of many types of information which will improve the performance of the four principle monitoring technologies -- seismic, hydroacoustic, infrasonic, and radionucleic -- at various regions of interest around the world. The number of different data sets in the KBase is expected to be huge, but viewed from an information representation perspective, there are only four basic types of data: reference event information, parametric grid information, contextual information, and supporting information (Shepherd et al., 1998). This report focuses on the access and use of the second of these types.

Parametric grid information is *interpolatable* data associated with particular geographic locations (i.e. a grid) on the earth. The interpolation part of the definition is important: not all data sets which have geographic coordinates should be interpolated. Examples of proper, interpolatable parametric grid data sets are traveltime corrections for any of the sensor technologies. The grid of locations for which information has been specified may be regular or irregular, depending on the method of collection or generation of data. Regardless, the KBase information derived from these grids must meet the USNDC operational requirements, i.e. the interpolated values must: 1) include a robust error estimate; 2) be first and second order continuous; and 3) be delivered to applications "quickly" (the precise speed requirement depends on the application).

In this report we describe in detail a general method for preparing, storing, and accessing interpolatable information which can insure that these requirements are met for any data set. The technique meets the requirements by combining the error estimation properties of kriging with the fast and robust interpolation properties of Natural Neighbor Interpolation. To clarify the use of the technique, we provide an example using a synthetic travel time correction data set, and show how use of the resulting KBase data set provides improved locations and appropriate error estimates.

(This page intentionally left blank)

2 Parametric Grids in the Knowledge Base

The end-to-end model (Figure 1) for inputting raw data to create a parametric grid data set and using it to output interpolated values to an application consists of three basic parts: *data preparation (DP)*, *data storage (DS)*, and *data access(DA)*. The first two, which we collectively refer to as *Knowledge Base population*, are one-time setup processes which are not required to be fully automated and, if necessary, may be computationally expensive. This is not true of DA, which involves serving the KBase data requests from client applications, often with stringent performance requirements. KBase access must meet the performance requirements for these requests, which in many cases imply sub-second response times. Thus, DA processes must be very fast.

We do not discuss raw data collection here, as we assume that data sets have already been assembled by contributors to the KBase. Each interpolatable raw data set *must* include for each data point a geographic location (latitude, longitude, and depth), a value, and an estimate of the error or variance associated with that value. In addition, if possible, an estimate of the spatial range of influence of each point should be provided (see section 2.1.1 below). Though an important topic, we also choose not to discuss the specifics of client applications that interface with the functional implementation of this research in order to limit the breadth of the document. We do provide a brief description of the Application Program Interface (API) to the Knowledge Base, libKBI, in section 2.3.1 below.

In the remainder of this first part of the document, we discuss in greater detail DP, DS, and DA, respectively, breaking each down into sub-processes as appropriate. In the DP section, we discuss the kriging interpolation method and associated modifications for generating continuous value surfaces from raw data. In addition, we describe the tessellation method whereby the continuous value surface is converted to a discrete mesh for purposes of achieving high performance numerical interpolation. Finally, we discuss the mesh refinement process by which the generated tessellation is made to optimally represent the continuous kriged value surface as accurately as desired with as few discrete nodes as practicable. In the DS section we define the process by which new data and their resulting refined mesh value surfaces are merged with existing data in the KBase in cases where regional overlaps occur. This section also describes the types of data stored by the KBase and the underlying relationships between the data types. In the final section, DA, we cover the interface used by client applications to access data from the KBase as well as the actual methods and strategies used to perform the data retrieval. This section defines the major points of the discrete triangle searcher and the Natural-Neighbor Interpolator which are provided in the client application interface.

2.1 Data Preparation (DP)

DP is required because for many data sets it is not possible to quickly produce interpolated values with high quality error estimates directly from the raw data

points. Fast interpolation methods such as *Natural-Neighbor Interpolation* (NNI, described below) can provide interpolated values quickly but have no robust mechanism for error estimation. Error limits can be interpolated just like the values, but unless the spacing of the raw data is very small and/or the gradient of the actual surface is gentle, these linear estimates will be poor. Kriging methods (discussed in the following sections) provide a convenient means to interpolate at any point and get a high-quality error estimate reflecting the spatial correlation characteristics of the underlying process, but, when many data points are included, they are too slow to be used in fulfilling the real-time data acquisition requests routinely made of the KBase.

To meet both the robust error and speed requirements, we use a hybrid technique in which we employ a modified kriging method to produce a densified set of points with value and error estimates which then serves as a basis for NNI, a basis which is sufficiently dense where needed to insure good value and error estimates. The optimal basis for NNI should be a set of points whose density distribution is proportional to the surface curvature of the data process which was sampled in the raw data set. Choosing this set ensures that any errors associated with the linear interpolation are small. As we shall see, the optimal basis should also have the points arranged so as to form well-behaved *Delaunay triangles* (Delaunay, 1934). Delaunay triangles are required to properly perform the NNI.

Generally, speaking DP consists of two main activities. The first, *surface generation*, is used to provide sampling at additional locations besides the raw data points. As previously mentioned we use a *Modified Bayesian Kriging* (MBK) technique to generate these value surfaces. The second activity, *mesh generation*, is used to specify the connectivity, or *tessellation*, of all of the data points which is required for searching for natural neighbors and for calculating the weights of the neighbors for interpolation. When performing mesh generation, the tessellation process uses the values of the kriged surface in an iterative manner to refine the final grid into an NNI mesh that optimally represents the kriged surface to within a prescribed tolerance using a minimal set of grid points. This combined process is called *mesh refinement*. In the next few paragraphs we shall consider each activity -- kriging, tessellating, and mesh refinement -- in order.

2.1.1 Kriging

For many KBase clients it is not sufficient to get the best available estimate of a parameter, we must also know how accurate each estimate is thought to be. For example, in the case of interpolating travel time correction data, the corrections will change the location regardless of their accuracy, so it is essential to know the error associated with the corrections to evaluate the new location. If error information is provided by the KBase, it can be incorporated into the error ellipse routine associated with the location algorithm and thus be reflected in the size of the error ellipse associated with the new location.

$$Z_p^*(\hat{X}_p) = \sum_{i=1}^{n} w_i Z_i(\hat{X}_i) \qquad \text{(EQ 1)}$$

where $\hat{X}_i$ and w_i are the position and weight of the ith point, respectively. Kriging has two important features that distinguish it from other linear interpolation methods, however. First, in kriging the weights are determined such that the variance of the predicted error is minimized. Second, and of critical importance for our purposes, in addition to the estimated value at the unknown point, kriging also produces an estimate of the error associated with the interpolated value.

To properly krig a specified data set using the MBK method, one should first perform a *variogram analysis* of the data to determine the data's spatial correlation characteristics. Next, the variogram analysis is used to establish the data's *covariance characteristics*. Finally, the data covariance is input into the kriging algorithm, along with the raw data and associated data measurement errors, and the kriging operation is executed to generate continuous value and error surfaces, i.e. values and error estimates at any location.

In its standard form, kriging assumes that the underlying processes are spatially invariant and that the non-spatially correlated error (i.e. measurement error) associated with each data point is the same. Both constraints are problematic for our purposes and so we use a modified version of the algorithm which allows them to be removed (Schultz et al., 1998). In the modified version, each point can have a unique measurement error and the modeled processes can be spatially variant. The MBK method is discussed in section 2.1.1.4 below and in greater detail in Appendix A.

2.1.1.2 Variogram Analysis

To krig a data set, one should first perform variogram analysis of the data to determine the spatial correlation parameters. To do this for a given data set, we first create an *experimental variogram* and then fit it with a *theoretical variogram*, which will ultimately be used in the kriging. The experimental variogram, $\Gamma(h)$, is made by calculating the squared value differences for all combinations of points, binning these by interpoint distance h, and then plotting the results as a function of h, i.e.

$$\Gamma(h) = \frac{\sum_{i,j} [Z_i - Z_j]^2}{2N_{ij}} \qquad \text{(EQ 2)}$$

where N_{ij} is the number of pairs of i,j points in the h distance bin, and we have written Z_i as a shorthand for $Z_i(\hat{X}_i)$ for compactness. Technically, if the 1/2 factor is used the result is a *semi-variogram* (multiplying by two gives the variogram). However, the difference is not important so long as one or the other is used consistently, and often the term variogram is used for either. Regardless, once the experimental plot has been made, the theoretical variogram is determined by fitting the trend of the

data with a mathematical function which is described with four parameters: the *nugget*, the *sill*, the *range*, and the *function family* (see Figure 3). Variogram analysis is a topic unto itself, and is covered at length in many geostatistical text books (see e.g. Kitanidis, 1997).

In theory, the variogram should stabilize to some constant value (the sill value) with distance as the limit of spatial correlation is reached (the range). If the geometric distribution of the points has relatively few long inter-point distances, however, the variogram may not stabilize and the sill value and range may be difficult to determine. This is because the variogram bins at large distances may have too few observations to form a representative average value. To prevent mis-interpretation, we recommend *always* comparing the experimental variogram with a plot of the number of distance pairs per bin. Bins with small numbers of samples should be disregarded if possible.

Determination of the nugget value can be equally ambiguous if the point set has relatively few short inter-point distances. The nugget value is caused by the presence of *random (i.e. non-spatially correlated) error* in the data. Any portion of the values of X_i due to such a process will only correlate when comparing a point with *itself*; for comparisons with all other points no matter how close, the error will not correlate. Thus the effect on the variogram is to introduce a step discontinuity between 0 distance and the first finite distance. The nugget value, which is the variance (i.e. square) of the non-correlated error, is defined by the apparent intercept of the theoretical variogram with the Y axis, ignoring the drop off at zero distance (Figure 3). Accurate determination of this intercept is dependent on the number of small inter-point distance combinations. In many data sets, this information may come from a single cluster in which case one should use caution in generalizing the results as the nugget for the entire data set, though there may be no better alternative. If the X_i have very little measurement error then there will be no discernible discontinuity and the nugget will be zero.

The choice of the function family is perhaps the most subjective of all of the theoretical variogram parameters. Any mathematical function which fits the basic trend between the nugget at zero distance and the sill at the range distance *could* be acceptable, which leads to an infinite number of possibilities ranging from a simple linear fit to a much more detailed fit such as a spline. Fortunately, there are a few guidelines to follow in choosing a function family. First, unless the experimental variogram has highly resolved detail which clearly supports the use of a complex function, it is better to use a simple one. Using a simple, smooth function will lead to smoother, simpler kriged surfaces. Also, complexities in the function shape may lead to instabilities in the matrix inversion required for kriging. Second, functions with zero or near-zero slope at small distances can lead to kriging instabilities for some data sets. Such functions imply perfect or near-perfect correlation over some finite range of distance. If the nuggets (i.e. uncorrelated errors) for clustered data points are small and the observed values at the points are dissimilar, the theoretical correlation will be incompatible with the observations and problems may occur. In such

cases, we recommend either shortening the range of the zero slope portion of the variogram, increasing the slope, or both.

To summarize, the fitting of an experimental variogram with a theoretical variogram is a highly subjective process wherein the researcher must find parameters which fit but do not overfit the data, and which do not lead to instabilities in the kriging process. Adhering to the simple guidelines prescribed above will generally produce satisfactory results.

2.1.1.3 Variograms And Covariance

For reasons we shall discuss below, we choose to work with covariances rather than variograms. Fortunately, the relationship between a theoretical variogram and the covariance is straightforward:

$$\Gamma(h) = cov(0) - cov(h) \qquad \text{(EQ 3)}$$

where $\Gamma(h)$ is the variogram at distance h, cov(h) is the covariance at distance h, and cov(0) is the covariance at distance 0. The transformation is illustrated in Figure 4. Note that the nugget leads to a discontinuous decrease in covariance (variance) beyond zero distance. The covariance value at the first finite distance beyond zero is equal to the variogram sill value minus the nugget value.

2.1.1.4 Modified Bayesian Kriging

As already mentioned, we use a non-standard version of kriging when performing DP. Our method is a modified version of *Simple Kriging* (i.e. kriging with known mean, Wackernagel, 1995) which we call *Modified Bayesian Kriging* (Schultz et al., 1998, see also Appendix A). MBK follows the Bayesian Kriging methodology developed by Omre (1987). MBK is based on Simple Kriging rather than the more typical Ordinary Kriging, because we assume that our data has been demeaned and hence that we know the mean of the underlying process (i.e. it is zero). By using Simple Kriging, we control the extrapolation behavior of the kriging, guaranteeing a return to zero values away from data points. The assumption of zero mean also implies that before the data can be kriged the underlying model must be removed to detrend the data. For example, for regional travel time data, the data to be kriged would be the observed residuals for ground truth events relative to a regional travel time model. If the mean were not known, Ordinary Kriging could be used but this would increase the error estimates for interpolation points due to the uncertainty inherent in determining the mean (Wackernagel, 1995). For MBK it is always assumed that the mean is known and equal to zero. This constraint of zero mean allows us to derive the kriging equations without imposing a bias constraint, so MBK is not truly a BLUE process (rather it is a BLE process). This is not without precedent in the kriging literature, however, and others have noted that in some cases removing this constraint can improve results (Cressie, 1993).

The MBK modifications to Simple Kriging address two shortcomings of the basic kriging technique which were problematic for our purposes: the assumption of constant measurement error for all data points, and the lack of a means to include apriori constraints on the range of influence for each point. We discuss the MBK approach to each of these below.

2.1.1.4.1 Data Point Dependent Measurement Error

MBK allows *data point-dependent measurement error* (i.e. uncorrelated error). Measurement error is often incorporated in kriging (this is the nugget, as we discussed above), but it is generally assumed to be the same for all data points. We know that for many of the typical KBase data sets the error associated with each data point can vary considerably (e.g. earthquakes vs. nuclear explosions), so this is a poor assumption for us. In order to allow the specification of point dependent measurement error, we work with covariances, variances, and correlation coefficients, rather than the traditional variogram function. Using Equation 3, we can derive these quantities from the variogram, as shown in Figure 4. For example, for P travel time corrections for station ABC, we first make a variogram of all the P residuals relative to the base model. If we have separate estimates for the measurement error associated with each point, these are used for kriging. Otherwise, the measurement error (i.e. square root of variance) for all points is taken as the square root of the nugget. The theoretical covariance value at the first finite distance beyond zero is equal to the variogram sill value minus the nugget value. In the terminology of MBK, this is the background variance. It is the assumed zero distance covariance for any theoretical point which we krig for (these points do not have measurement error). The correlation coefficient function is obtained by normalizing the covariance function, with the nugget removed

There may be some confusion regarding the covariance function as explained here. Notice that it is a function of *interpoint distance only.* Thus the meaning of the value of the covariance at a given distance is that it should reflect the correlated variance of all possible point combinations with that distance, i.e. it *does not* refer to correlations of the variance due to multiple realizations of the value at two distinct points as might be expected from the traditional statistical definition of covariance. Such a derivation is impossible in this case because we have only one sample (i.e. data point) at each location; additional realizations are not available. This is an important distinction to make and if understood should prevent confusion regarding the use of the covariance information in kriging. Essentially, the total variance realizable at any theoretical point is the same, and is equal to the covariance at zero distance (i.e. variance) less the nugget value. How this total variance correlates with that at another point determines the covariance between the points. To get the covariance function, we multiply the total variance by the cross correlation function. Thus, the value of the covariance function at any distance reflects *two factors*: the total realizable variance at the two given points separated by that distance, and the correlation of the two points. Perfect correlation with no variance at each point (i.e. a flat surface) will lead to zero covariance as will zero correlation with large variance

at each point (a high-amplitude white noise surface). Note, however, that the last example will have a large covariance at zero distance (i.e. variance) -- this is the nugget.

2.1.1.4.2 Apriori Range of Influence

The second MBK modification allows the use of apriori range of influence information for each point. For data sets with good spatial sampling, this is not necessary because any underlying processes should be defined by the data itself. For many typical KBase data sets, however, the spatial sampling is poor and the analyst may have additional information that cannot be derived from the data due to the limited sampling (e.g. the scale length of a tectonic province). Our modification allows the incorporation of some apriori information by associating a radial blending function (1 to 0) with each data point X_i, i.e.

$$Z(X_i) \rightarrow B_{ip} Z(X_i) \qquad \text{(EQ 4)}$$

where B_{ip} is a function of the distance between point X_i and the kriged point X_p. Basically, what this does is limit the influence of data points in the interpolation as they move further away. It can be thought of as a *smoothed version of Neighborhood Kriging*. However, it has the advantage of avoiding the discontinuities associated with Neighborhood Kriging, because the weights associated with points that move into or out of the kriging neighborhood fade in and out smoothly rather than changing discontinuously as a point leaves the neighborhood.

In the MBK equations, this blending function multiplies the correlation function and thus can only act to decrease the range of correlation (see Schultz et al., 1998 and Appendix A, EQ. A-22). This behavior makes MBK behave much better as an extrapolator when interpolating in regions away from data point sampling, as intended, but it can prevent proper interpolation in regions with good data point sampling. For this reason, MBK is implemented with options to control the use of blending in areas with good sampling. At present, we provide two options which inhibit blending dependent on either the density of surrounding points or the azimuthal coverage of surrounding points. In either case, the blending is only used when the point in question is deemed to be isolated from surrounding points. When the point is in a region of good data sampling there is no blending and the equations essentially reduce to Simple Kriging with zero mean. Further, the transition between the two cases is implemented in a smooth fashion such that no discontinuities are created.

In the paragraphs above we have defined the MBK process including how to prepare the data by removing trends and assigning measurement errors and blending functions for each point, and how to perform a correlation assessment using variogram analysis. The functionality to perform the variogram analysis and subsequent MBK calculations are implemented as Matlab m-files and are available from Sandia

Labs upon request. Details of the variogram analysis and kriging functions are given in Appendices E-1 and E-2, respectively.

Now that we have examined how a parametric grid surface is generated, we are ready to consider the second part of the DP task, mesh generation or tessellation. Recall that mesh generation is used as a framework for representing the global kriged surface as a discrete high performance and locally generated interpolated surface. The necessary tessellation framework is the subject of the next section.

2.1.2 Tessellation

With kriging, we can calculate values with error estimates at any set of points in space. The set of points that we choose to represent the kriged surface by, and how those points are connected, is commonly referred to as the *tessellation*. The KBase calculates and saves *Delaunay tessellations*, a well characterized and intensely studied tessellation whose connectivity is specified by triangles. The tessellation is then used by the KBase to rapidly find the nearest node neighbors of any arbitrary location, which are needed for NNI. Given those neighbors, the tessellation is used to calculate the weights necessary to perform the NNI. The process of evaluating the proper tessellation, however, is actually done in parallel with the kriging process in such a way that a minimal number of nodes is used to represent the kriged surface as an NNI surface to within an arbitrary pre-defined level of accuracy. This process is referred to as *mesh refinement* and will be discussed later in section 2.1.3 and in much greater detail in Appendix B.

Proper implementation of a Natural-Neighbor Interpolator requires that a Delaunay tessellation be used to form the node-to-node connectivity information (Delaunay, 1934). Delaunay tessellations, which for our purposes are triangles on 2-D or 3-D surfaces, are of particular interest because they minimize the maximum angles (or inversely maximize the minimum angles) of the triangles created in the tessellation. Fortune (1992) refers to this property as the maximum-minimum angle property. This property tends to form equilateral triangles and leads to configurations where the average node valence (the number of edges connected to a node) is approximately six. Generally speaking, well proportioned triangles fair better with regard to minimizing local data variability and numerical errors accumulated during the interpolation calculation.

In addition to forming well-shaped triangles, Delaunay triangle sizes are largely determined by the density of the nodal distribution from which they are created. This makes the Delaunay tessellation an ideal selection for tessellating sparse and scattered data where nodal density is proportional to the surface curvature of the data set. This is precisely what is obtained with the mesh refinement technique to be described later.

Many algorithms exist that perform Delaunay tessellations. Some of the more popular algorithms include "edge-flipping" (Fortune, 1992), "incremental insertion" (O'Rourke, 1994), and the convex hull algorithms such as the Barber, Dobkin, and

Huhdanpaa (1993) "quickhull", to name a few. For purposes of constructing the KBase parametric grids we have implemented an *incremental insertion* algorithm. We chose this algorithm for several reasons. First, it is easily modified to use the current connectivity of the mesh, prior to the next node insertion step, to find the triangle that contains the node to be inserted. Using spatial locality this way, to find the insertion point of the next node, removes much of the performance degradation seen by most incremental methods that don't implement a built in searcher. Second, the NNI weight calculation assumes that the point to be interpolated is a new data point to be added to the tessellation which is functionally equivalent to the incremental insertion algorithm. This permits a large subset of code used to perform tessellation construction to be reused during the interpolation calculation. Third, incremental methods allow for easy insertion (and removal) of nodes, locally, into existing tessellations which means that mesh modification is greatly simplified. Modifying an existing mesh simply means the nodes are incrementally inserted (or removed) into (or from) the existing mesh within a small local connectivity region. The method does not require that the entire tessellation be rebuilt from scratch. Finally, an incremental approach can easily be modified to tessellate a spherical surface, which is the surface (the Earth) that the data is ultimately represented upon.

The incremental Delaunay tessellation algorithm at any point in the mesh construction is as follows:

For each new node to be inserted

1. Find all triangles whose circum-circle contains the new node (a circum-circle is a unique circle that passes through the three nodes of its triangle).
2. Remove each triangle whose circum-circle contains the new node to form an enclosing polygon that contains the new node.
3. Connect each node of the enclosing, or "insertion" polygon to the new node to form new triangles that all share the new node.

This process is illustrated in Figure 5. Figure 5a shows the original tessellation and the new node to be inserted. Figure 5b highlights the triangles whose curcum-circles contain the new node. Figure 5c depicts the "insertion" polygon after the containing triangles are removed. Finally, Figure 5d illustrates the new tessellation after the new node is connected to each node of the insertion polygon.

Using this process, an entire tessellation can be built from an arbitrary collection of nodes. The next question is what is the collection of nodes from which the tessellation should be assembled? More specifically, how are the nodes defined to utilize the initial raw data, represent a kriged surface to a prescribed level of accuracy, and simultaneously minimize the number of nodes required to form the tessellation? This process, known as *mesh refinement*, is the subject of the next section.

2.1.3 Mesh Refinement

The goal of mesh refinement is to produce a minimal or "sufficient" basis for performing fast NNI that produces values within a prescribed error bound of the kriged dataset surface. But what points should comprise the tessellation to ensure that the accuracy requirements are fulfilled? An obvious choice is a regular grid over latitude and longitude with the spacing small enough to capture the smallest scale variations that have been sampled in the underlying data process. Such a grid would certainly provide a sufficient basis to insure high-quality results from the NNI, but it would also include many superfluous points in areas where the kriged surface varys little over a large range of grid points. These extra points will lead to degradations in performance when the node and associated data are accessed from the KBase and when the natural neighbors are found before performing the interpolation. A much better solution is an irregular grid with grid spacing proportional to the surface curvature of the sampled underlying process. The irregular grid need have no more points than are required to insure a specified level of accuracy in the NNI.

We use an iterative method to find the *optimal irregular grid* for each data set. Initially the raw data points are inserted into a coarse background grid and the grid is checked for accuracy and roughness to see if it is a sufficient basis for NNI. If it is not, new intermediate points are added and the process is repeated until a sufficient basis is found. The iterative process following mesh initialization can be defined by five distinct steps:

1. **Tessellating**

   Any new refinement nodes found in the previous iteration are added to the existing tessellation using an incremental method (The first pass creates the initial tessellation from the raw data and a pre-defined coarse background mesh).

2. **Smoothing**

   Each of the new data points, excluding the original data and boundary nodes, are smoothed using a length-weighted Laplacian smoother to remove poorly formed triangles;

3. **Kriging**

   Every new data point is assigned a correction by performing an MBK calculation for each new triangle node;

4. **Curvature Refinement**

   All nodes that exceed surface roughness/curvature requirements are split by creating new nodes to be inserted at the mid-points of the edges that share the rough node;

5. **Accuracy Refinement**

   Lastly, all remaining triangle edge midpoints and centers are interpolated using a *gradient-modified version of NNI* and kriged using MBK and differences are formed. New nodes are created for insertion into the tessellation at all edge and triangle mid-points for which the differences exceed a user-specified relative error tolerance between the interpolated and kriged values (A transition to absolute error is made for data values that approach zero).

Steps 1 through 5 are repeated until all relative error and surface smoothness requirements are satisfied. The mesh refinement process and data flow is illustrated in Figure 6. A detailed discussion of the mesh refinement process is given in Appendix B. The gradient-modified NNI method is discussed later in the DA section (section 2.3) and is described in detail in Appendix D. The necessary mesh refinement implementation functionality is coded in Matlab m-files and C++ source, available upon request, and is described in Appendix E-3.

Note that the kriging, tessellating, and interpolating processes are fundamentally inter-twined during mesh refinement: new nodes are inserted into the tessellation at points where the relative difference between the kriged and NNI results differ by more than a prescribed amount. Thus, the existing tessellation is continually updated as new points are added near spatial locations that did not meet the relative error requirements.

Once the optimal NNI grid has been determined, the information must be stored in the DS step. This is the subject of the next section.

2.2 Database Storage (DS)

2.2.1 Merging with Existing Global Data

When a researcher chooses to contribute parametric grid data to the KBase, that information must be merged with any existing information of the same type. For example, a KBase contributor might deliver a grid of P travel time corrections for a given station for a given region that overlaps with existing KBase corrections for the *same* station/phase pair for another region. In fact, such an overlap is guaranteed for all but the first contributor of a given data type because all the parametric grids stored in the KBase are *global*. Rather than attempting to merge the contributed grid with the existing global grid -- a difficult process which can lead to many problems (e.g. discontinuities at edges) -- we instead add the ground truth data points from the new grid to those for the existing global grid and *regenerate* the global grid through mesh refinement. The resulting mesh should be a cohesive, continuous representation of the interpolation of the combined ground truth data points. Thus, smooth transitions between study regions are assured. Also, the global grid can be readily re-generated at any time so editing of any of the data points by the contributors or the addition of new data points is easily handled.

2.2.2 Data Storage

Once the data have been reconciled with the global set and a global parametric grid has been determined that accurately represents the kriged value and modeling error surfaces, the information (tessellation and associated value surfaces) can be entered into the KBase. Currently, the storage mechanism is a simple binary flatfile, but the actual storage mechanism for operational use is still undecided. In this document we shall not discuss the details of the storage mechanism and concentrate, instead, on the type of data that is stored in the database.

The primary types of parametric grid data that must be stored in a database regardless of format include:

1. Generic header information including the date of creation, the version numbers of the kriging and mesh refinement software used to generate the data, and the maximum tessellation, basemodel, station and phase ID's used to construct the surface identifier hashing function;
2. Basemodel, Station, and Phase names and associated ID's used to construct surface identifiers (see paragraph below for a definition of a 'surface');
3. Surface accounting information including the number of surfaces, and for each surface, the surface id, the basemodel, station, and phase names from which the surface was formed, the back-ground modeling error used by the surface, and the tessellation id of the tessellation upon which the surface is represented;
4. Tessellation accounting information including the number of tessellations, and for each tessellation, the tessellation id, the number of nodes, triangles, and represented surfaces, and the tessellation's modified-gradient parameter flag;
5. Mesh data for each tessellation including a description of all nodes that comprise the tessellation and the connectivity of the tessellation in the form of triangle information. Each node is defined by an ID, a location that includes the longitude, latitude, and depth of the node, and a type flag that indicates specific node type information. Each triangle is defined by an ID, a set of three node ID's that are the triangle's vertices, and a type flag that gives additional triangle specific information;
6. Correction data defined for each surface at each node of the surface's owning tessellation. This data includes a correction value, a longitudinal and latitudinal derivative, and a modeling error value (this is travel-time specific... other data types, such as hydro-acoustic or infrasound, may have different data storage requirements).
7. Mesh refinement calculation setup and flag information which is necessary to reliably regenerate the mesh data that is contained within the database; and
8. Kriging calculation setup and flag information which is necessary to completely regenerate the kriged surfaces from which the refined mesh was evaluated.

In the list above the concept of a *surface* is introduced which refers to the actual stored data defined at all nodes of a specific tessellation for a unique identifying

basemodel, station, and phase set. Each surface is owned by a single tessellation. That is, each surface only uses one tessellation's node-set to tie its values to. Surfaces are *not* multiply defined across many tessellations. However, it *is* possible to include more than one tessellation in the data storage. Each tessellation has its own node and connectivity (triangles) definition and owns some subset of the surfaces defined in the database.

In the next section we shall discuss how the libKBI interface software uses stored data to provide requesting client applications with interpolated results.

2.3 Data Access (DA)

Once data have been processed and stored, they are ready to be accessed and used by client applications. The access and use of the data consists of three primary steps: *data retrieval* from the KBase, *containing triangle search* via the walking triangle method, and the *NNI calculation* to return results. Each of these functional steps are executed by the *libKBI software* which serves as the implementor of the Data Access process.

This section will begin by discussing briefly the functional interface of libKBI. Subsequent sections will describe each of the three data access steps including data retrieval, which defines the methodology and format for extracting data from the KBase; the containing triangle search which refers to the searching method used by libKBI to find the client requested interpolation point within the tessellation(s) connectivity structure; and, the NNI including the gradient-modification used to ensure that the NNI is differentiable at all node points.

2.3.1 libKBI Functionality

Parametric grid data retrieval and subsequent processing is handled by a library of C++ KBase interface routines known as libKBI (library Knowledge Base Interface). Specific function calls can be inserted into client source code to make requests from the libKBI API during execution. The libKBI functionality can be divided into six specific tasks including:

1. database connection,
2. surface fetch initialization (SFI),
3. data cluster fetch size initialization,
4. surface association,
5. location initialization, and
6. surface interpolation

The first task, *database connection*, is simply a call to establish a connection to the KBase through a database manager such as Oracle, SDE, or even a flat-file rep-

resentation. This call is a standard request providing the names of the server, database, password, and other pertinent information required to make a proper connection.

The second task, *surface fetch initialization* (SFI), refers to the manner in which DA is restricted, or enveloped, to include only a subset of the data available in the KBase. DA is restricted to optimize data retrieval performance by minimizing the amount of data that must be retrieved from the KBase and stored in-core. This is generally accomplished by eliminating data (surfaces) from retrieval consideration if they are not necessary for the current problem defined by the client application accessing libKBI. As discussed above, the concept of a *surface*, used here and throughout the remainder of section 2.3, refers to the data that comprise traveltimes, magnitudes, amplitudes, etc., and their derivative and modeling error values for a specific basemodel, station and phase assignment. The concept of a surface, as well as the methodologies and software abstractions used to enable the data access restrictions are discussed in the next section on *data retrieval.*

The third task, *data cluster fetch size initialization*, defines another restriction on the amount of data from particular surfaces that can be loaded at one time. The *data cluster* is best viewed as a spatial cookie-cutter that only retrieves data from requested surfaces whose nodes lie within the boundaries of a pre-defined polygon (the cookie-cutter). This concept is also described in detail in the next section.

The fourth task, *surface association*, allows restriction of which surfaces are actually interpolated to a subset of those that were loaded based on the SFI prescription. Like the previous two tasks, this one is also described in more detail in the next section.

The fifth task, *location initialization*, simply sets the longitude and latitude of an interpolation request from the client application. Once libKBI receives this information it is able to perform a containing triangle search which is prerequisite to calculating the interpolated values at the provided location. The containing triangle search is discussed briefly in section 2.3.3 and in greater detail in Appendix C.

Finally, the sixth task, *surface interpolation,* refers to the actual NNI calculation at the current location specified by the client. The NNI calculation is described below in section 2.3.4 and in much greater detail in Appendix D.

2.3.2 Data Retrieval

We have just seen that libKBI handles all issues associated with retrieving data from the KBase. Specifically, it accomplishes the data retrieval by restricting the amount of information that is loaded in-core and by further restricting interpolation calculations on a surface-specific basis. The amount of data retrieved is restricted in two different ways. First, the available surfaces in the KBase are culled to a predefined subset as defined by the client application. Second, a spatial data cluster (cookie cutter) is used to further restrict the amount of data loaded to that which lies

within the boundary of the data cluster. We shall describe each of these methods of restricting the accessed and processed data in the paragraphs below. But first we need to define the abstract representation of a tessellation object to fully understand how libKBI accesses the data and makes it available for use by a client application.

2.3.2.1 Tessellation And Surface Object Abstraction

In section 2.2.2 we discovered that the KBase, regardless of implementation, must store surface, node, and triangle information within the database. We also saw that the KBase was capable of storing many tessellations and that each tessellation, in turn, could have many surfaces assigned to it. Figure 7 illustrates this concept abstractly with an imaginary KBase that includes 4 tessellations. Each tessellation contains the nodes and triangles that define spatial location and connectivity information for that tessellation. In addition, each tessellation contains one or more surfaces that utilize the spatial information of the tessellation (the nodes) as locations to define its values on. For example, assume that "surface 1" is travel-time information which consists of a single travel-time value for each and every node defined within the tessellation. The assembly of all such values over their respective nodes forms a discrete surface. Using an interpolation function one can define a continuous interpolated surface over the entire range of the tessellation.

In our imaginary KBase of Figure 7 we see that "tessellation 1" has 4 surfaces, "tessellation 2" has 5 surfaces, "tessellation 3" has 2 surfaces, and "tessellation 4" has 3 surfaces. Notice that the index number of each surface is unique regardless of which tessellation owns the surface. This is an intended data requirement of the KBase. The KBase does not allow a surface to be represented by more than one tessellation. One can imagine the trouble that would undoubtedly occur if libKBI tried to access the surface assigned to say basemodel 'AK135', station 'CMAR', and phase 'P' only to find that the surface index existed in two different tessellations. Which one would be used?

The relationship between a tessellation and the surfaces that it owns can be made clearer by examining Figure 8. Figure 8 shows constituent parts of "tessellation 4" of the imaginary KBase. The parts of a tessellation consist of geometry information (nodes and triangles) and value information (surfaces). Notice that all geometric spatial location information is provided by the nodes of the tessellation. A node is, quite simply, the latitude, longitude, and depth of some point on the earth. In and of itself it has no other attributes assigned to it. The triangles, on the other hand, are a means of tracking the mesh connectivity of a tessellation. A triangle is simply the indices of each of the three nodes that comprise the triangle. The node indices are always ordered in a counter-clockwise fashion relative to an outward pointing normal on the surface of the Earth. Triangle connectivity provides a means for determining node neighbors and is vital for performing the containing triangle search and the node neighbor calculation both of which are required to perform NNI.

The actual data values that are to be interpolated are contained within the surface object representations. In Figure 8 three surfaces are shown; 12, 13, and 14. Notice that all three use the tessellations' nodes as the spatial location upon which their values are defined. The only difference between one surface and the next is the magnitude of the value at each of the tessellation's fixed nodes and not its location.

2.3.2.2 Surface Fetch Initialization (SFI)

Now that we have examined the abstract nature of the tessellation object relative to how it is represented in libKBI we are ready to investigate the three primary data retrieval tasks of libKBI. Recall that the first data retrieval task (task 2 in section 2.3.1 above) was *surface fetch initialization* (SFI). The purpose of SFI is to define a specific set of surfaces that are defined as *necessary and required* by the calling client application to properly perform its intended function. By restricting the number of surfaces accessed by libKBI to a subset of those available in the KBase a substantial performance gain may be noticed. Also, computer resources (memory) may be in short supply and minimizing the amount of in-core data can help to alleviate this problem.

Figure 9 depicts a surface fetch subset from the imaginary KBase. In the example presented in Figure 9 surfaces 1, 2, and 3 of "tessellation 1", surfaces 5, 7, and 9 of "tessellation 2", surface 10 of "tessellation 3", and surface 14 of "tessellation 4" are designated for access while surfaces 4, 6, 8, 11, 12, and 13 of their respective tessellations are restricted from access. In this example we have reduced the number of surfaces required for access by almost *50%*. In reality the number of surfaces may be in the thousands while the fetch subset is generally in the single or double digits. *Real problems can see savings of a factor of 10 to 100 in access performance.*

As an example of the use of SFI, consider the location of a seismic event with a small subset of the station/phase combinations available in the KBase, e.g. only P phases for the stations at regional distances. Once the event is ready to be located, the relevant station/phase combinations are known, so SFI can be used to limit the scope of the queries to the KBase in the data access step and thereby improve the speed of the KBase access..

2.3.2.3 Data Cluster Access

The next data retrieval task performed by the libKBI API is *data cluster fetch size initialization* (task 3 in section 2.3.1). Defining a data cluster fetch size is also a method of minimizing the amount of data that must be acquired and stored in-core. Recall that the data cluster was best viewed as a spatial cookie-cutter that only retrieves the requested surface data from nodes that lie within the confines of a predefined polygon (the cookie-cutter). The need for this type of functionality may provide the best reason to use a sophisticated commercial product for DS rather than a

gle on the list would be accessed and subsequently checked to see if it were the containing triangle. This process would continue until, ultimately, the containing triangle of the current interpolation location was found. Unfortunately, although easy to implement, brute force methods such as this are computationally expensive if the number of triangles is large. In fact, these methods on average have search times that are proportional to 1/2 the number of triangles loaded from a tessellation.

Other grid and tree based techniques can be used that tend toward search times that are proportional to the square root of the number of triangles but they require additional storage structures beyond that already required to represent the tessellations connectivity. Instead, we chose the *Walking Triangle Algorithm* (see Sambridge, 1995 or Lawson, 1997) as it also achieves square root performance but doesn't require additional storage structures to implement.

In this technique, an initial triangle is selected and tests are made to find whether the current client supplied location of interest lies to the left or right of each edge of the triangle. Each edge of the triangle is treated in succession. If the point falls to the left of an edge, the next edge of the triangle is subjected to the same test. If the point falls to the right, however, the search shifts to the triangle to the right and the process is repeated. When finally a triangle is found for which the point lies to the left of all three edges, the containing triangle has been found, and the process can continue on toward determining the interpolated value at the requested location. The result of a typical triangle walk process is illustrated in Figure 13.

Generally, the first time a triangle walk occurs for a specific event location, the first triangle in the currently loaded list of triangles for a specific tessellation is used as the initial guess. On subsequent requests, however, the previously determined containing triangle is used as the initial guess for the next iteration. This has a decided advantage from the point of view of typical event location clients in that they tend to issue interpolation requests that progressively "home-in" on the correct event location. By choosing the previous containing triangle for the initial guess of succeeding searches the total number of triangles that must be traversed to the new location is always minimal.

The Walking Triangle Algorithm works equally well in a flat plane or on the surface of a sphere as implemented in libKBI. The only modification is minor and concerns the determination of the "leftness" or "rightness" of the edge tests when processing the search. The details of the algorithm for both flat plane and spherical surfaces are provided in Appendix C.

With the containing triangle determined for a client-requested event location we are ready to proceed with the NNI. This is the subject of the next section.

2.3.4 Natural-Neighbor Interpolation (NNI)

At this point we are ready to consider the methodology behind NNI. However, some background may be informative on why NNI was chosen from an exhaustive

list of potential interpolators. Many interpolation methods are available for general use and all vary with how well they fit arbitrary data representations. Generally speaking they can be organized into the two distinct classes of *fitted functions* and *weighted average methods*. Fitted function methods generally require that a set of coefficients to some polynomial be determined by solving a related set of linear equations involving the data and constraints or criteria that control the fit of the function. The fitted function approach summarizes data behavior in a global manner. With few exceptions, fitted function methods are typically poor models for sparse data distributions in that the resulting surface can behave unexpectedly; over- and under-shoot between data points being a typical problem.

Weighted average techniques, on the other hand, sum the data influences at an interpolation point from all neighbor points that lie within the interpolation points *influence* region. Weighted average techniques are typically local in nature and allow local surface trends to be captured which, generally speaking, is not always possible with the fitted function approach. We were forced early on to consider essentially local methods since they only required a small neighborhood of associated data to accurately perform the interpolation calculation. The reason for this concerns the shear volume of data contained by the KBase which was seen to be much larger than might possibly be loaded into core memory at any one time. If global data were required to perform a surface interpolation it could become prohibitively slow as the quantity of surface data for individual surfaces grew larger. In fact, the kriging interpolation method uses this global calculation methodology which eliminated it from further consideration for our fast interpolator (even though it would have been simpler to implement a single interpolation scheme for the entire DP through DA process).

Of the many available weighted average interpolation techniques, NNI appears to best fit all of our requirements. These requirements include:

1. The ability to model an existing surface accurately,
2. Possession of first and second order continuity and differentiability,
3. Dependence on local neighborhood data only, and
4. Properties exhibiting strong surface stability regardless of data density and anisotropy (scattered and irregular).

For our purposes, NNI can always accurately model a surface by simply densifying the tessellation using mesh refinement as discussed in section 2.1.3. It also is continuous and differentiable everywhere except at the actual tessellation node points where the differentiability criteria fails. However, a *small modification* (gradient-modification), which is described below, remedies this inadequacy quite simply and easily. As we shall soon see, NNI depends on the smallest neighborhood of local data and thereby satisfies the third requirement. Lastly, NNI always exhibits stability and furthermore never propagates erroneous data values beyond its small influence neighborhood. This can't be said of many weighted average schemes or any

fitted function schemes. Watson (1992) gives an excellent survey of the major types of computational interpolation methods employed today, including NNI, and highly recommends NNI as a sparse-local data interpolator.

Natural-neighbor interpolation, as discussed above, is a local weighted average interpolation method. These methods (in two dimensions) have an interpolation function of the form

$$f(x,y) = \sum_{i=1}^{n} w_i(x,y) f_i \qquad \text{(EQ 5)}$$

where i is summed over all node neighbors that surround the interpolation point (x, y), f_i is the function value at the nodes, and $w_i(x,y)$ are the normalized *influence* weights attached to neighbors (each i) that influences the interpolation result at (x, y). In NNI these weights are referred to as the natural-neighbor coordinates of point (x,y) (Watson, 1992). Our treatment simply refers to them as weights. A brief, qualitative synopsis of the method by which the natural-neighbor weights are calculated will be given here. Appendix D covers the method in more detail, including the gradient-modification necessary to ensure first and second order differentiability at the node points. Refer to Watson (1992) and Sambridge (1995) for excellent treatments of the full procedure.

The natural-neighbor weights are best described using a *geometric* definition. Examine the five nodes in Figure 14. The dashed lines connecting the nodes represent the Delaunay triangles whose circum-circles include the interpolation point (see section 2.1.2 for a description of Delaunay triangles and circum-circle containment). The solid lines are the voronoi edges of the nodes, the complete set of which is referred to as a voronoi polygon (Preparata, 1985). Each node in a Delaunay tessellation has exactly one convex voronoi polygon that completely surrounds it. The voronoi polygon can be constructed by passing a perpendicular line through the midpoints of each edge of the Delaunay tessellation. The intersection of those lines form the vertices of the voronoi polygon. In Figure 14, we have removed outlying edges for clarity.

Figure 14 depicts the result of adding the interpolation point to the tessellation. The shaded portion of Figure 14 represents the new voronoi polygon for the interpolation point. The new voronoi cell about the interpolation point overlaps all of the original cells of its natural-neighbors. The natural-neighbor weight for a neighbor node acting on the interpolation point is defined as the ratio of the area of overlap with the neighbor nodes original voronoi cell and the new voronoi cell to the total area of the new voronoi cell. Since the nearest-neighbor weights are normalized they always lie between 0 and 1. This results in the property that if an interpolation falls exactly on top of a node the function value of the node is returned. The other important property to note is that the interpolation is entirely local and only depends on the nearest-neighbor set of the interpolation point.

This type of interpolation is linear and results in a surface that has continuous first order derivatives throughout the tessellation except at the data points themselves. A completely first and second order continuous interpolation can be evaluated by blending the linear form with the natural-neighbor gradient evaluated at the interpolation point. Watson (1992) describes some simple parametric blending functions that account for the linear interpolant, gradient, and the variability in the data to control the surface tautness. Accounting for the neighbor gradient at the interpolation point is termed the *gradient-modified* (GM) NNI. The method is described in detail in Appendix D.

Recall that the natural-neighbor weights must be calculated for each unique tessellation as determined by the surface association task of libKBI (section 2.3.2.4 above). Once the weights are known they are multiplied by the surface values (the f_i in equation 24 above) and summed to obtain the interpolated value at the requested client location. The appropriate weights of each tessellation multiply all "event associated surfaces" within their domain. Examining Figure 12, once more, will help to clarify this statement. Given the weights for "tessellation 1", they will multiply the values in surface 3 to obtain the interpolation value at the client location for surface 3. Similarly, the weights from "tessellation 2" will multiply values from surfaces 7 and 9, and weights from "tessellation 4" will multiply values from surface 14. The interpolation process proceeds this way each time the client requests an interpolated value for a specific event associated surface.

2.4 Summary

This completes the discussion of *data preparation* (DP), *data storage* (DS), and *data access* (DA) of parametric grids within the KBase. To review, DP begins by examining the relevant raw data for each surface to be stored in the KBase using *variogram analysis* to determine the spatial correlation behavior of the data, from which kriging parameters can be derived. With this information, a *tessellation* will be created that minimizes the number of nodes needed while maintaining a predefined level of accuracy between the kriged surface and the NNI surface which will be used to represent it in the KBase . This occurs in the *mesh refinement* step of DP. Next, in DS, the NNI surfaces are *merged* (actually re-kriged) with any existing data already present in the KBase and are stored. Finally, in DA, when requested by a client through *libKBI*, data from the KBase is extracted using a *surface fetch initialization* (SFI) prescription to restrict the number of accessible surfaces and a *representative data cluster size* to restrict the amount of data accessed per SFI surface. This set of data is further restricted from interpolation processing on a per event basis by the client through a *surface association prescription*. The remaining surfaces are processed to find the *containing triangles* of all locations as prescribed by the client applications. Then an NNI calculation is performed and all interpolated values are returned to the client for each of the final surfaces defined in the event's *surface association set*.

To help clarify these concepts, in the next section we provide an example using seismic travel time correction data.

3 Example: Seismic Travel Time Correction Data

Historically, the organizations producing global earthquake bulletins have used 1D radial Earth models to locate events (e.g. the ISC uses or did use JB -- Adams et al., 1982). Improved accuracy is obtained by applying a series of corrections to account for non-radial effects: the elliptical shape of the Earth, the non-standard elevation of individual seismic stations, near-receiver velocity structure, and event-dependent path velocity variations (Figure 15). Note that the order of the corrections is important because each assumes that the others have been applied before it, i.e. the corrections are relative to other corrections. The ellipticity and elevation corrections are easily calculated and nearly always applied. The near-receiver term (known as the "bulk static correction") is less common, but simple to calculate once a large number of residuals (after ellipticity and elevation have been corrected for) are available. Path corrections are the least often applied, but have long been recognized as essential when particularly accurate locations are needed.

Calculating path corrections for repeated events from the same site is simple enough -- the correction per station/phase pair is calculated from historic events and applied to any future events from the same site -- however, interpolation of the corrections for new locations is less straight-forward. Past approaches have generally involved fitting a smoothed surface to the observed residual data: e.g. a second order polynomial surface (an "SRST") or a bicubic spline surface (an "SSSC"). While these surface fitting approaches are simple and easy to understand, they suffer from two significant problems. First, the surfaces are smoothed fits (see the discussion of fitted-functions in section 2.3.4), which means they fit only the average trend of the data, regardless of the accuracy of the data. Thus, in the case where some of the residual points have greater accuracy than others (e.g. well-recorded nuclear tests vs. earthquakes) there is no way to make the fitted surface follow those points more closely. Second, the smoothed surfaces have no error associated with them. Essentially, once a surface has been fitted, the connection to the data is lost; every point on the surface has equal confidence. This second problem is of critical importance for CTBT monitoring because verifying a possible violation will require not only the best possible location but also an accurate estimate of the location *error.*

For large, well-recorded events, path residuals tend to average out so path corrections are generally unnecessary and the error associated with them is irrelevant. To successfully monitor a CTBT, however, requires detecting and locating small events which are generally recorded by much fewer stations (a 1 kt well-coupled event is equivalent to mb 3.8 - 4.5; Lay, 1997). Further, these stations tend to be at regional distances where global (or even regional) one-dimensional models typically do not fit the data as well. For such events, if the error associated with the model is not properly accounted for in the location algorithm, then more than (100 - X)% of the time the X% confidence ellipses calculated for many smaller events may not contain the true locations, as was noted for regional PIDC events detected and located in the first few years of operation. The situation at the PIDC has been improved by updating the global one-dimensional models to include estimates of "modeling error"

(though in fact this is actually bias). In a statistical sense this has "fixed" the problem because the error ellipses are now larger so that they generally contain the true event locations, but in fact the actual amount of mislocation has not decreased.

The preferred solution, of course, is to reduce mislocation *and* the size of the error ellipse where possible based on the available data. As we shall see, we can accomplish this using the parametric grid capability described in Section 2 to store and apply path correction information. In this case, the raw data points are the locations of historic seismic events and the values to be interpolated are the residuals relative to a travel time base model (which may include corrections for ellipticity, elevation, and near-receiver structure). The correction surface will fit each point as tightly as is warranted by the measurement error specified for that point, and each interpolated value will include an error estimate based on the statistics derived from the ground truth points themselves. Using a synthetic data set, we demonstrate this process and show how it results in improved event locations and error ellipses.

3.1 Data Set

For our test data set we created a set of 121 synthetic events in North Africa recorded by a set of 3 IDC primary network stations: CMAR, GERES, and ZAL (Figure 16). These stations were deliberately chosen to poorly constrain the event locations to better illustrate the importance of using path correction information; they in no way reflect the most likely stations to detect events from this area. Our events span a rectangular grid of 10-30° N. latitude, 10-30° E longitude, and are evenly spaced every 2 degrees. For each event, we first created IASP91-consistent P arrivals (with ellipticity and elevation corrections) for each station. Then for each event, we perturbed the arrival time to each station using an order 50 spherical harmonic surface whose coefficients were randomly generated, and which had been scaled to have values of up to +/- 2 seconds (Figure 17). We used a separate surface for each station, and the perturbation value for each event was taken from the value of the surface at the true geographic location of the test event. In effect, the surfaces represent the path-specific information not fit by the base model (i.e. the effects of the lateral heterogeneity of the Earth). The use of spherical harmonic surfaces as opposed to purely random noise insures that the perturbations are tied to an underlying process with *spatial correlation characteristics*, as would be expected were the perturbations due to actual Earth structure.

This set of perturbed events should have the proper characteristics for demonstrating the KBase parametric grid capability. The observed times are inconsistent with the global one-dimensional model which will be used to locate them, and so must be corrected to achieve improved locations. Further, because the perturbations come from known analytic functions, we can compare the interpolated correction values with the actual values and the error estimates to evaluate the effectiveness of the interpolation process and the validity of the error estimates.

3.2 Re-Locations without Path Corrections

We begin by re-locating each of the events using the IASP91 base model with ellipticity and elevation corrections, but no path corrections. For our first case, we do not use any a priori error information and thus the 95% error ellipses reflect only the travel time residuals for the final event locations. This is no longer consistent with operational practices at the IDC, but is useful for illustrating the importance of including a priori error estimates in the calculation of the error ellipses. The results are shown in Figure 18. In general, the mislocations increase from NE to SW as the events move further outside of the aperture of the 3 station network (all three stations are located in the NE quadrant). The largest mislocation is 270 km for the event at 10 E, 10 N. The shape and orientation of the error ellipses are controlled by the network as well, though they are generally so small that they are covered by the event location symbols. If they could be seen, it would be apparent that their size increases following the same NE to SW trend as do the mislocations. However, the error ellipses are much too small (the largest is only 44.6 km) because we have not accounted for the model error associated with using the IASP91 model for our perturbed data. For 52 of the 121 events (43%), the 95% ellipse does not contain the true event location. Clearly, this is an unacceptable result.

Next, we repeat the relocations using a priori error information. We assume picking (i.e. measurement) errors of 0.1 seconds for all arrivals, and use the distance-dependent IASP91 modeling errors included within the IASP91 travel time tables used at the IDC (Walter Nagy, personal communication). The model error has a constant value of 0.70 s for epicentral distances from 18-112 degrees, which includes all of our source/receiver combinations, hence for these events model error does not change with source location or station. We calculate 95% *coverage* rather than *confidence* error ellipses (i.e. we assume an infinite number of degrees of freedom) to incorporate the a priori error information (see Jordan and Svedrup, 1981). In the location code the measurement and model error are combined to form one a priori error term for each arrival. The relocation results are shown in Figure 19. The locations are the same as in the previous case because the data has not changed and since the model errors are identical for each station, the relative weighting has not changed. The size of the error ellipses has increased dramatically, however, and now only 5 out 121, or 4%, of the 95% ellipses do not contain the true event location. Thus, we see that using the IASP91 modeling error estimates has made the error estimates *consistent* with the true mislocations, which is a significant improvement, though mislocations have not decreased.

3.3 Ground Truth Events

Let us now suppose that we have available 30 ground truth points for our region. For our test case, we have randomly selected 30 points from a 1 degree spacing grid spanning our area of interest (Figure 20). For each of these events the travel time residual relative to the base model for each station is just the perturbation value

from the spherical harmonic surface for that station. We can use these ground truth points and the associated values and error estimates to construct a KBase parametric grid data set which can be used to generate interpolated path corrections for any location using the methodology described in Section 2.

Before doing this, it is important for the reader to understand that the parametric grid data set created will be a sufficient basis to match the *kriged surfaces* derived from the 30 ground truth points, which are not necessarily the same as the *true spherical harmonic surfaces* (e.g. Figure 17). The extent to which a kriged surface matches a true surface is dependent on the sampling provided by the ground truth points; if we have no ground truth points sampling a feature of the true surface, then we certainly cannot expect the kriged surface to match that feature. One should expect a close match near the ground truth points and a less reliable match further away from the points, which is, as we shall see, just what the error surface derived from the kriging indicates.

In the following sections, we follow the example data set through DP and DA processes (DS is not discussed). Refer back to the corresponding parts of Section 2 for details.

3.4 Data Processing

3.4.1 Variogram Analysis

We begin with *variogram analysis*. For each of the stations we formed experimental variograms for the 30 ground truth points. The variograms for all three stations, as well as plots of the number of point pairs vs. interpoint distance, are shown in Figure 21. Variograms are notoriously ambiguous, as can be seen even in this case where we have used synthetic, noise-free data. We "eyeball" fitted the data and decide on the following parameters for the theoretical variogram for each: CMAR -- nugget = 0.05 s^2, range = 5 degrees, sill = 0.60 sec^2, curve family = butterworth; GERES -- nugget = 0.05 s^2, range = 5 degrees, sill = 0.40 sec^2, curve family = butterworth; ZAL -- nugget = 0.05 s^2, range = 5 degrees, sill = 0.70 sec^2, curve family = butterworth. Obviously these choices are non-unique; many other choices could be made which would fit the observations equally well and which could be used for *kriging*. Increasing the nugget will decrease the need to fit the ground truth point values and hence "smooth" the kriged surface. Increasing the range will extrapolated the values from the ground truth points further away, i.e it will extend the correlation range. Increasing the sill will increase the background variance (uncertainty) far away from all ground truth points.

3.4.2 Kriging

We will assume that we do not have any error estimates for the individual events and so we take the measurement errors for all residual points for each station from the square roots of the nugget values derived from the corresponding theoretical variogram. The nuggets are the same for each station here, but typically this would not

be true. We have no reason to limit the range of influence of any of the ground truth points, so we effectively "turn off" the blending by assigning a very large value (e.g. 99999 km) to the range of the blending function associated with each of the ground truth points. If desired, separate blending functions can be specified for each data point but in this case we use the same one for convenience. We use the same butterworth function family for convenience as well, though the choice of blending function family is unimportant due to the large range: the blending function will have a value of 1 out to (and beyond) the full range of the correlation coefficient function and so will have no effect.

Figure 22 shows the correction and variance surfaces obtained by using MBK for an 0.25 degree grid spanning our study area for GERES. Overall, the correction surface is not a close match to the spherical harmonic surface (Figure 17) due to the limited sampling provided by the ground truth points, but the match is good near the ground truth points as discussed above. The correction surface returns to zero value away from ground truth points because MBK is based on Simple Kriging with a zero mean (see Section 2.1.1.4). The variance surface shows deep wells around the ground truth points, and a smooth background level, as is typical for kriging. Note that the background level is 0.40 sec^2, and the variances at the ground truth points are 0.05 sec^2, as specified. Thus, the variance surface matches the information taken from the variogram for GERES in Figure 21, and listed above.

3.4.3 Mesh Refinement

Our next step is to use *mesh refinement* to create an optimal tessellation to use for NNI. We specify that the NNI result for any point must match the kriged value to within 10% and calculate the *optimal tessellation* for all three stations. We could calculate separate tessellations for each station but choose instead to form one tessellation that fits all three. This proves to be advantageous when the information is used operationally for event location because it requires only one walking triangle search to identify the natural neighbors needed to interpolate correction information for all of the stations as opposed to requiring separate searches for each as would be needed if each had a unique tessellation. Of course, this also leads to the presence of unnecessary points for each station, but this is easily offset by the gain in operational performance. Also, creating separate tessellations for each station actually leads to more point storage because each tessellation must be stored. Thus, we recommend combining multiple station and or phase information into a single tessellation whenever possible. A much more detailed discussion of single tessellations vs. multiple tessellations is given in Appendix B.

The tessellation process adds additional points in areas of high curvature where simple linear interpolation from distant neighbors does not work well. For the first station, we begin by imbedding the ground truth points within a coarse distribution of points surrounding our region of interest. The iterative process described earlier is then run until a tessellation is found which fits the error criteria. Regardless of the number of iterations and the amount of triangle smoothing which occurs, the ground truth points will always be included as nodes in the final tessellation (i.e. they are

stationary). Once the tessellation for the first station is finished, it is in turn used as the background for the next station. In our case, the second station has no new ground truth points, so no new fixed points are added, but this need not be the case; each station could have a separate set of ground truth points in which case the final set of fixed ground truth points would be the union. When the ground truth points are the same, the mesh refinement for each subsequent station tends to proceed more quickly because the areas of high curvature are near the ground truth points and so have already been finely meshed during the previous refinements.

The final tessellation and node density for all three stations for our examples is shown in Figure 23. The total number of nodes in this case is 2027. As suggested above, the densest areas of tessellation surround the ground truth points due to the curvature in the correlation coefficient function. The highest densities approach 100 nodes per square degree. If we had specified a smaller error tolerance (e.g. 5%) these numbers could be much larger. While we produce only one tessellation for all three stations, we produce separate value, error, and derivative surfaces for each. By surface, we mean the set of values which map to the nodes (see section 2.3). Theoretically, there is no limit to the number of surfaces which could be mapped to a single tessellation, though in practice it may make sense to limit them to similar types of data (e.g. all of the P corrections for the primary network) to keep the tessellations from becoming overly complex.

We examine the relationship between the true kriged surface and the NNI optimal tessellation surface in Figure 24. The top two panels show both surfaces for an 0.25 degree grid . Any differences are obviously too small to easily discern. To highlight the differences, we plot a difference surface in the lower panel. Note the dramatically changed color scaling: values range from -0.02 s to +0.02 seconds but most are much smaller. Recall that the kriged surface has deviations of up to +/- 2 seconds, so 10% error is 0.2 seconds. This better-than-specified matching is typical and is due to the fitting of multiple stations with the same tessellation. Each station adds points needed to match the kriged values for the residuals observed at that station, but because the points added for each station are not the same, the overall set of points includes additional points per station that lead to greater accuracy than was specified. Were we to have made an optimal tessellation per station, the misfits would more closely match the specified 10% error.

Once the tessellation and surfaces have been created, they can be stored and accessed using the libKBI interface.

3.5 Relocation Using Ground Truth Information

We relocate the events again, this time using the KBase parametric grid information we have created, accessed with libKBI. A few additional parameters are needed to run the libloc programs (LocSAT or EvLoc) using libKBI, and these are discussed in Appendix F. The re-locations are shown in Figure 25, and we have included the ground truth points for reference. The overall orientation (striking to the NE) and

trend (size increasing NE to SW) of the error ellipses has not changed, because they are determined by the network geometry. Superimposed on this however, are pockets of dramatic improvement clustered around the ground truth points. The overall average mislocation has decreased from 58.8 km to 47.0 km, but the most obvious improvements occur for events near (or on) ground truth events. The mean mislocation for the seven events co-located with ground truth points has been reduced from 42.7 km to 2.1 km. The reason that the mislocations are non-zero, of course, is that we assigned non-zero measurement errors to each ground truth event residual during the kriging, hence the kriged surface was not an exact fit. The size of the error ellipses for these events has also been considerably reduced, from a mean of 141.9 km to a mean of 50.9 km. These would be significantly smaller if the assumed measurement error for each test event pick were smaller (recall that we assumed 0.10 sec).

The sharp gradients in error ellipse size are due to the underlying kriging process (refer to the error surface for GERES in Figure 22) and are typical of KBase interpolated information. This is probably the most important result obtained from the new methodology. Using existing techniques to apply path correction information, the size of error ellipses would not have shown any correlation with the location of ground truth points because of the use of fitted-surface functions. The new behavior is more complicated but necessarily so to reflect the uneven spatial sampling provided by ground truth points.

(This page intentionally left blank)

3 Conclusions

The parametric grid capability of the Knowledge Base (KBase) provides a new, greatly improved method to store and access *interpolatable information* used in nuclear event monitoring. This method involves three basic steps: *data preparation* (DP), *data storage* (DS), and *data access* (DA). The goal of DP is to produce a *tessellated representation* of an associated *value and error surface* which can serve as a basis for the very fast *Natural Neighbor Interpolation* (NNI) in the DA step. In many cases, DP will involve the application of *Modified Bayesian Kriging* to *densify* the original data set such that the NNI will yield sufficiently accurate results. The densified information is stored in the DS step. Finally, in DA, values can be *interpolated* in near real-time by making a request through the *libKBI interface.*

Using a synthetic case of observed seismic travel times from a set of know (calibration) events, we have demonstrated the use of our method, starting from the input data and proceeding from DP to DA. We showed that both event mislocations and the sizes of the error ellipses are improved by using them. More importantly, we demonstrated that both the sizes of the mislocation and the size of the error ellipse are strongly tied to proximity to the calibration data, which must be true and yet has not been the case with past approaches. Those approaches have relied on using surfaces fitted to the correction information and as a result have produced overly smoothed correction interpolations with poorly determined error estimates.

While our example was limited to seismic travel time data, we fully expect that the technique can be applied to any of the other types of parametric grid data expected to be stored in the KBase: e.g. amplitude correction data, azimuth and slowness correction data. Though the "base model" for each of these differs greatly, the basic idea of interpolating correction information on top of each seems to be appropriate for all.

As a final word of caution, we note that while our method should greatly improve the accuracy of the various processes which use parametric grid information, this is only true if the underlying data is of sufficient quality. In particular, we note the paramount importance of establishing *robust error estimates* for each ground truth measurement. With the new methodology, the error estimate associated with each data point *strongly* controls the influence of that data point on the interpolation for nearby locations. Inaccurate error estimates will lead to either too much or too little influence for a given data point, resulting in an interpolation that is either incorrect or less accurate than it would have been with the proper estimate.

(This page intentionally left blank)

References

Adams, R. D., A. A. Hughes, and D. M. McGregor (1982). **Analysis procedures at the International Seismological Centre**. *PEPI*, **30**, 85-93.

Barber, B., D. P. Dobkin, and H. Huhdanapaa, (1993). **The Quickhull Algorithm for Convex Hull**, The Geometry centre technical report GCG53. The Geometry Centre, Univ. of Minnesota, Minneapolis, MN 55454.

Cressie, N. A. C. (1993). **Statistics for Spatial Data**, J. Wiley, New York.

Delaunay, B.N. (1934). **Sur la Sphere Vide**, Bull. Acad. Science USSR VII: Class. Sci. Math., 793-800.

Fortune, S. (1992). **Voronoi Diagrams and Delaunay Triangulations, Computing in Euclidean Geometry**, eds. Du, D.Z., and Hwang, F., World Scientific.

Hoppe, H. (1994), **Surface Reconstruction From Unorganized Points**, PhD Thesis, Department of Computer Science and Engineering, University of Washington.

Jones, R. E. (1979). **QMESH: A Self-Organizing Mesh Generation Program**, Sandia Report, SLA-73-1088.

Jordan, T. H. and K. A. Svedrup (1981). **Teleseismic Location Techniques and Their Application to Earthquake Clusters in the South-Central Pacific**, Bull. Seismol. Soc. Amer., 71, 1105-1130.

Kitanidis, P. K. (1997). **Introduction to Geostatistics**, Cambridge, Cambridge.

Lawson C. L., 1977, **Software for C1 surface interpolation**, Mathematical Software III, ed. Rice. J., Academic Press, New York.

Omre, H. (1987). **Bayesian Kriging -- Merging Observations and Qualified Guesses in Kriging**, Mathem. Geol., 19, 25-29.

Preparata, F. P., and M. I. Shamos, (1985), **Computational Geometry: an Introduction**, Springer-Verlag, New York.

Sambridge, M., J. Braun, and H. McQueen (1995). **Geophysical Parameterization and Interpolation of Irregular Data Using Natural Neighbors**, Geophys. J. Int., 122, 837-857.

Schultz, C. A., S. C. Myers, J. Hipp, and C. J. Young (1998). **Non-stationary Bayesian Kriging: a Predictive Technique to Generate Spatial Corrections for Seismic Detection, Location, and Identification**, Bull. Seismol. Soc. Amer., 88, 1275-1288.

Shepherd, E., R. G. Keyser, H. M. Armstrong, E. P. Chael, and C. J. Young (1998). **Data Dictionary for the CTBT Knowledgebase: Phase 1 Release**, Sandia Report #SAND98-1260.

Sibson, R. (1980). **A Vector Identity For The Dirichlet Tessellation**, Proc. Cambridge Philosphical Society, 87, 151-155.

Voronoi, Mg.G., (1908), **Nouvelles Applications Des Parametres Continus A La Theorie Des Formes Quadratiques**, J. Reine Angew. Math., 134, 198-287.

Wackernagel, H. (1995). **Multivariate Geostatistics**, Springer, Berlin, 256 pp.

Watson, D. F., (1992). **Contouring: A Guide to the Analysis and Display of Spatial Data**, Pergamon, ISBN 0-08-040286-0, 321 pp.

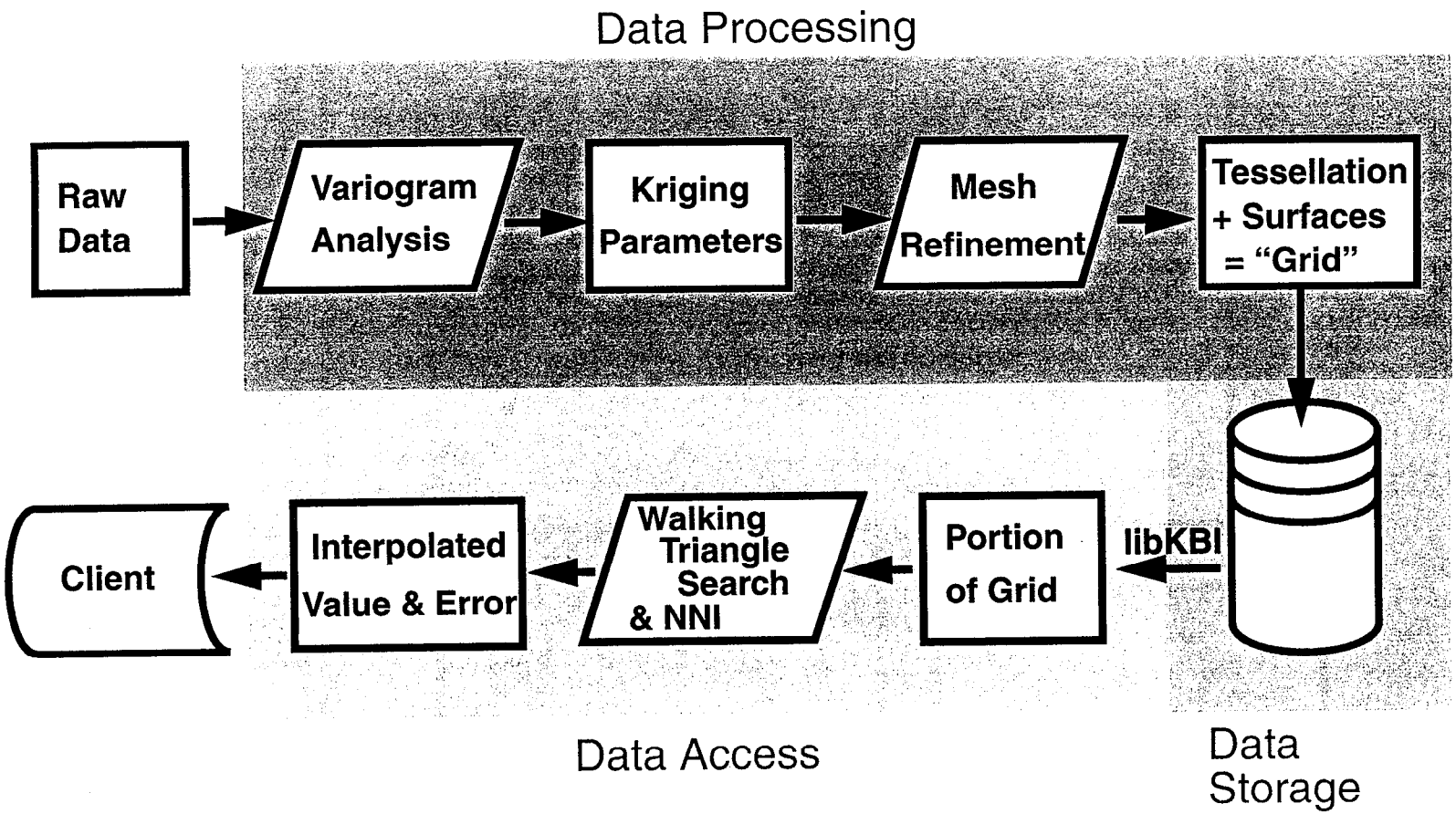


Figure 1. The Knowledge Base parametric grid model
The model shows how raw data is transformed into parametric grid information which in turn supports requests for interpolated information for KBase clients, such as an event location algorithm or a magnitude algorithm.

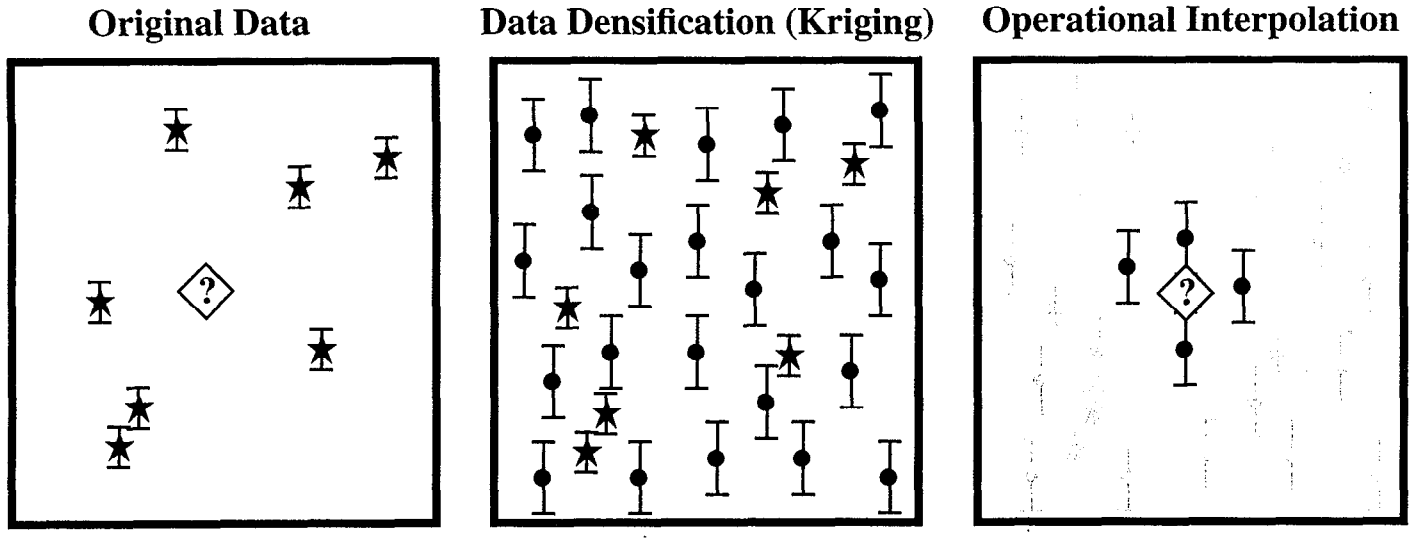


Figure 2. Densifying data for fast interpolation of value and error estimates
(Left) Original data -- The stars and error bars indicate raw data points,e.g. corrections to base model travel times for a given seismic phase between the indicated source location and a given station. The data are unevenly spaced, which may lead to problems with error estimates for interpolated points which are not nearby, such as the one indicated with the white diamond. **(Middle) Continuous data** -- Kriging can be used to "densify" the data set and produce robust error estimates for new locations so that any potential interpolation location will be surrounded by nearby points. **(Right) Interpolation** -- Combining the original data points and the new kriged data points gives us a more uniform spatial sampling, which allows us to use a simple interpolation scheme to quickly deliver accurate interpolated values *and* error estimates using only nearby points.

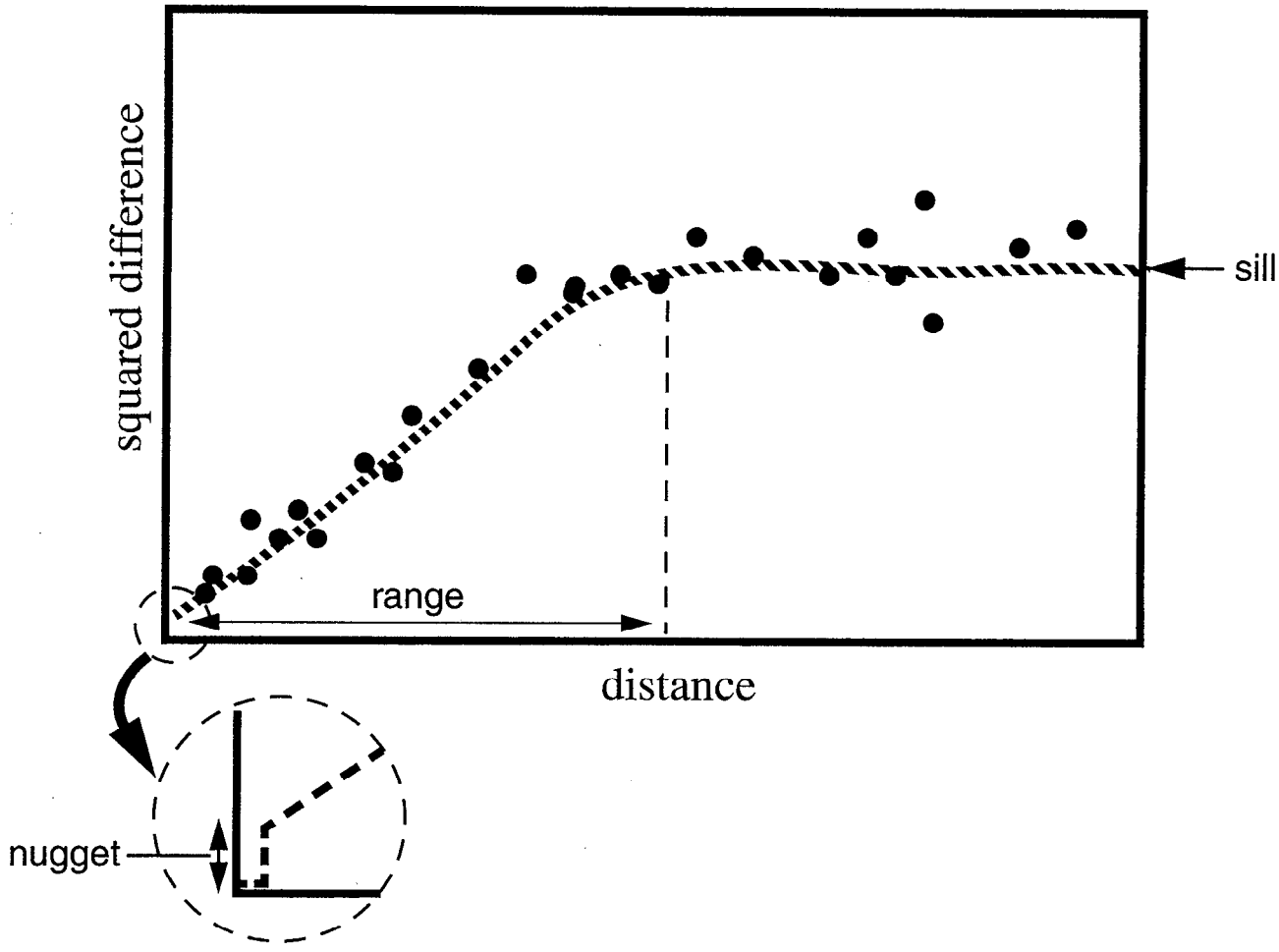


Figure 3. Observed variogram with fitted theoretical variogram
The observed variogram is created by plotting the squared difference for each pair of observations. The theoretical variogram is fitted to this, by focussing on the *nugget*, *range*, and *sill* values.

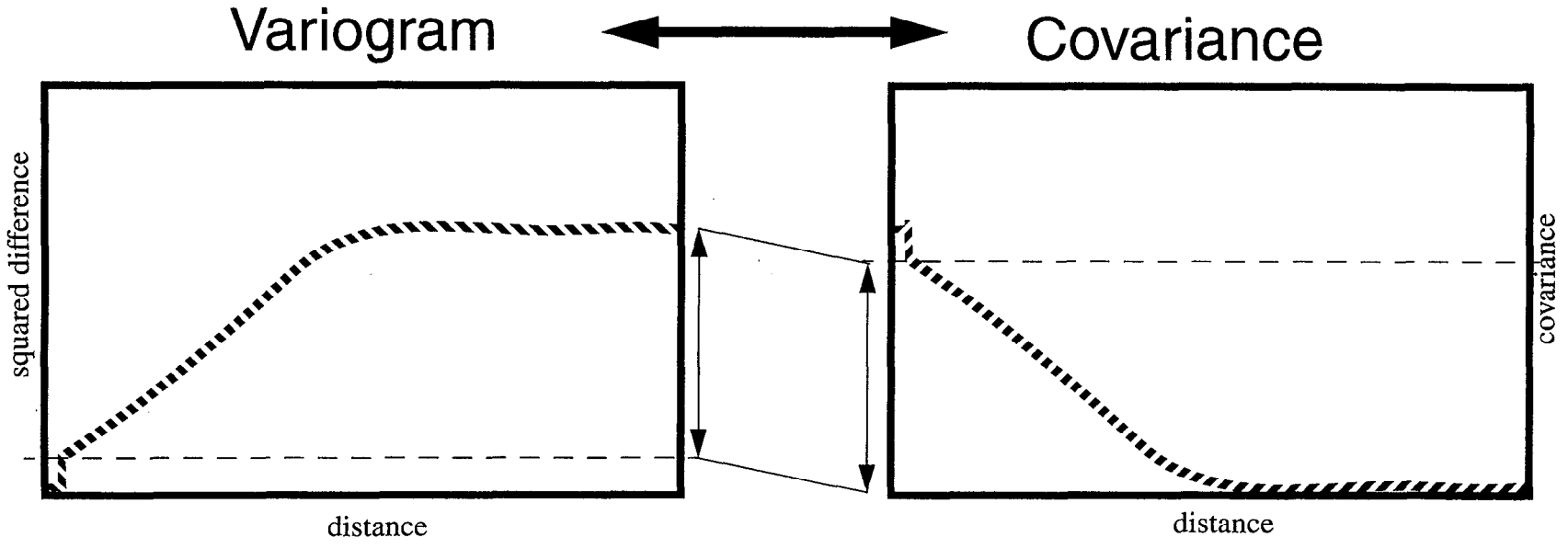


Figure 4. Variogram vs. covariance
The variogram is essentially the inverse of the covariance: increasing distance increases the variogram but decreases the covariance. Roughly speaking, one can equate the variogram with dissimilarity and the covariance with similarity.

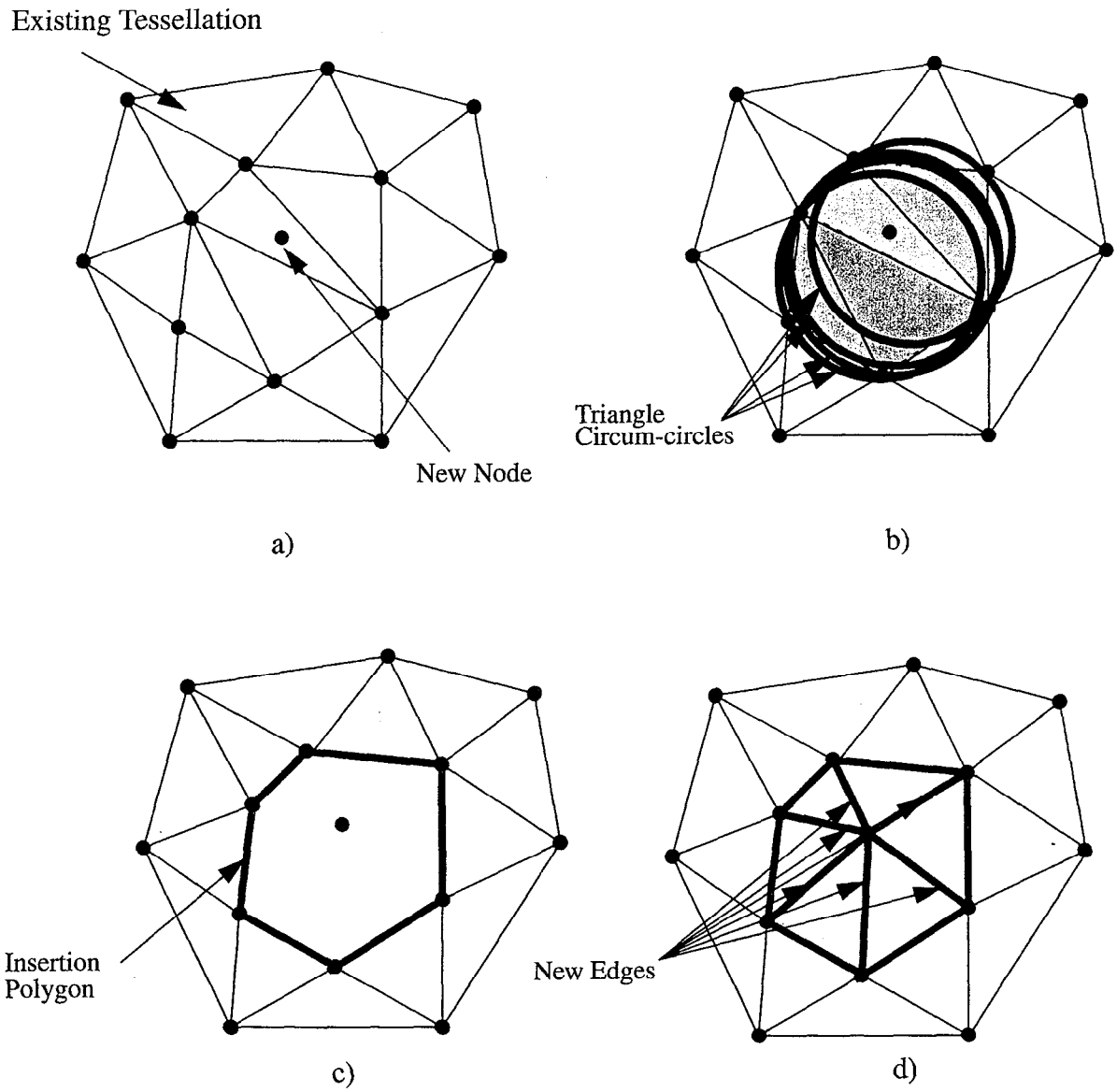


Figure 5. Incremental node insertion
a) Shows the original tessellation and the new node to be inserted into the tessellations, b) depicts circum-circle containment of the new node by each of the highlighted triangles, c) illustrates the "insertion" polygon after the containing triangles are removed, and d) shows the new tessellation after the new node is connected to each node of the insertion polygon.

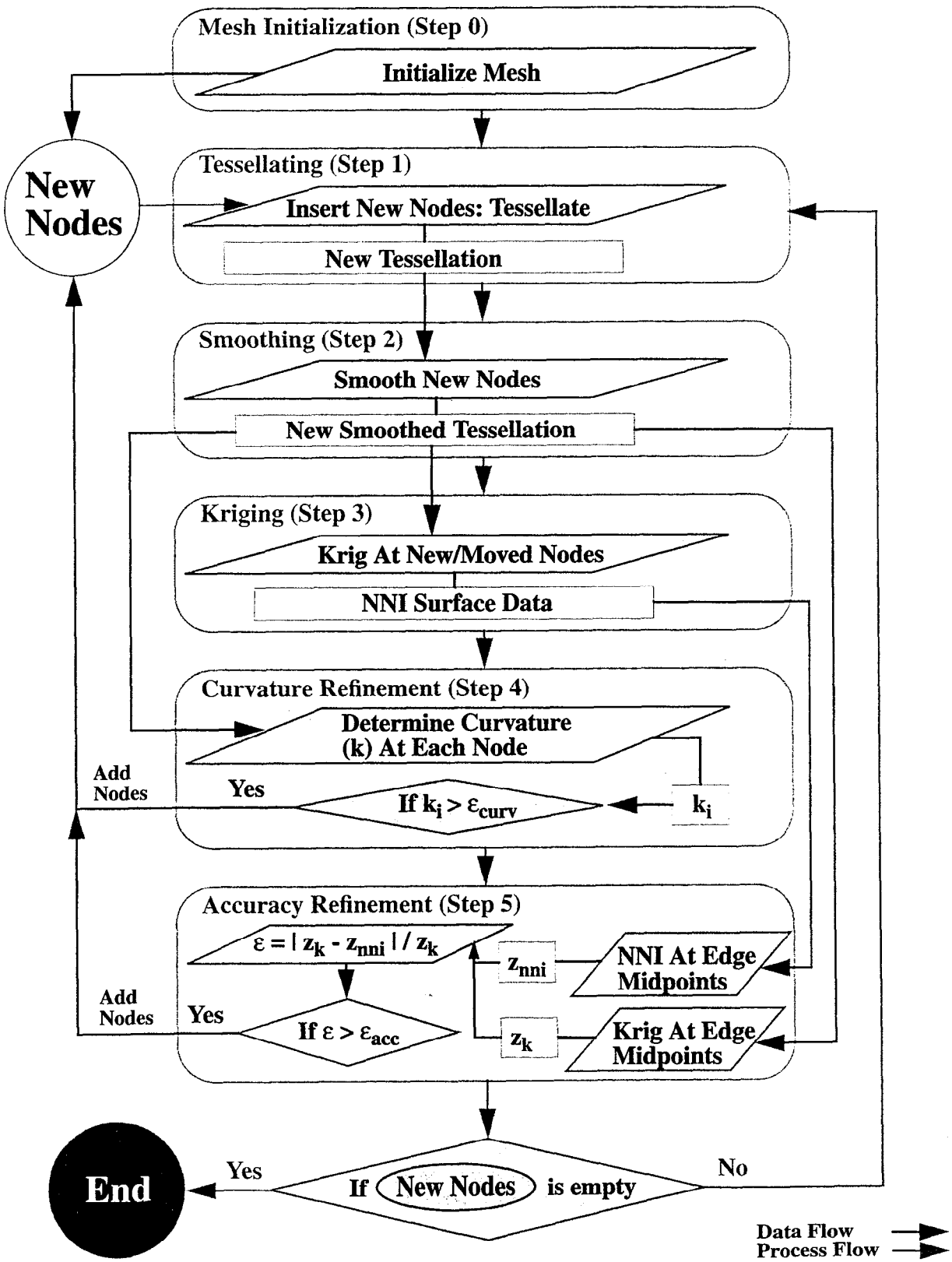


Figure 6. Mesh refinement process and data flow diagram

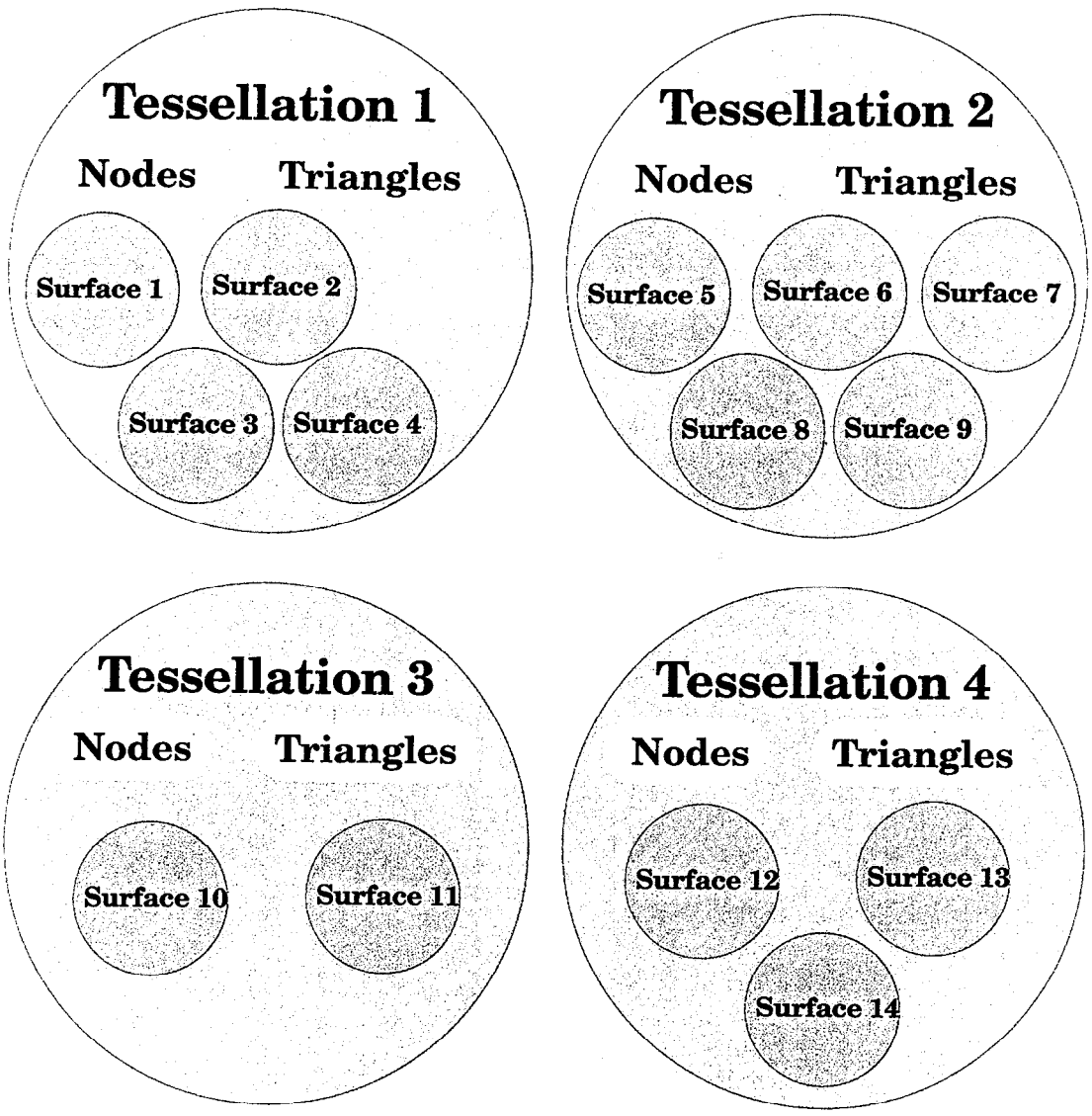


Figure 7. Abstraction of KBase storage for multiple tessellations
The KBase can store multiple tessellations each with it's own geometric definition (nodes, and triangles), which defines spatial location and connectivity for the mesh, and surface ownership, which contain the value surfaces created by the kriging/ mesh refinement process.

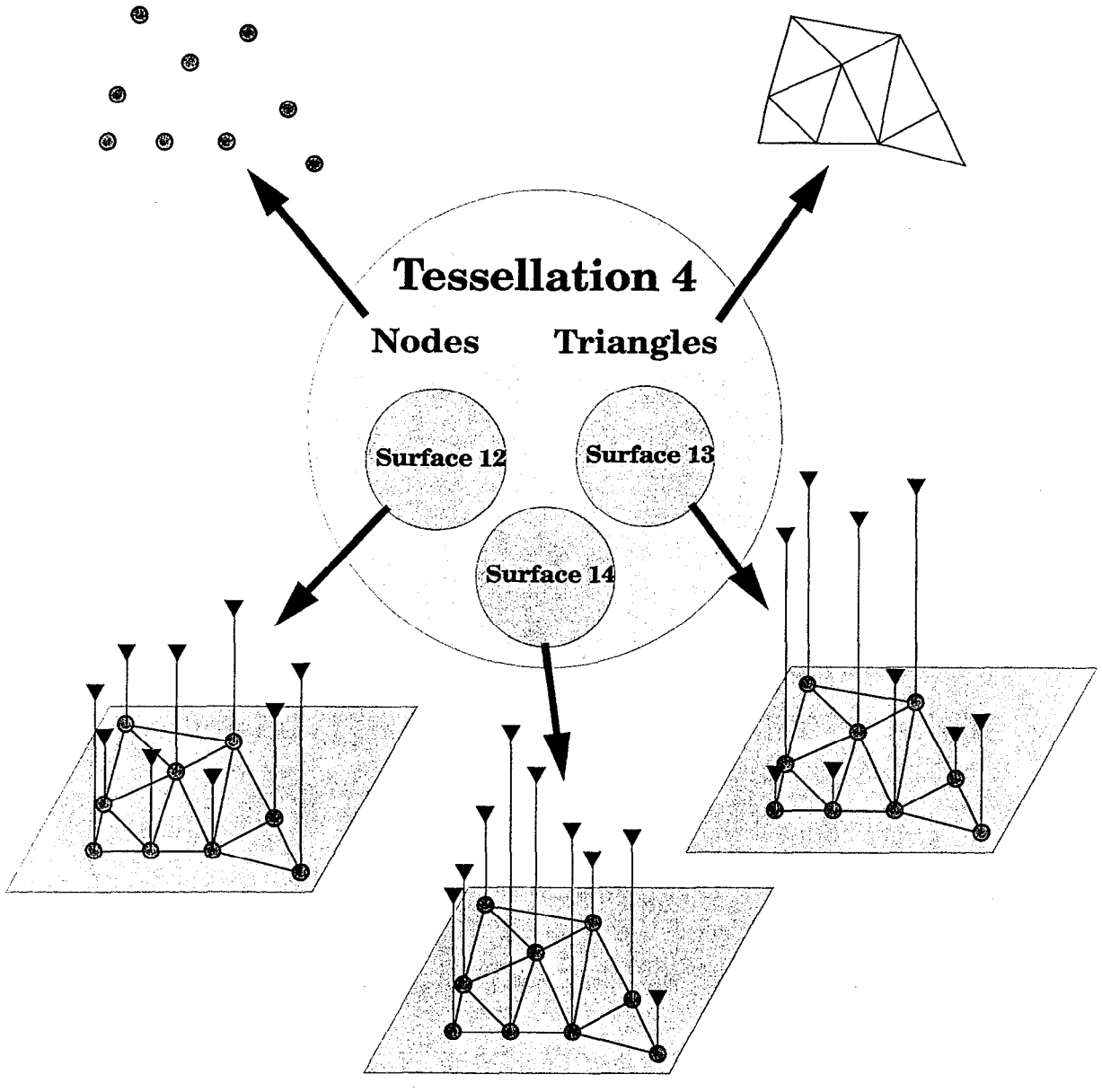


Figure 8. Abstraction Of KBase tessellation components
The KBase Tessellation contains geometric data that define the spatial location (nodes) and mesh connectivity (triangles) of the tessellation. The tessellation also contains an arbitrary number of unique surfaces that contain discrete values for each node in the tessellation. Each surface generally represents a unique basemodel, station, phase designation. The values can be any arbitrary parameter (or several of them jointly) such as travel-time, amplitude, azimuth, etc., and their associated surface derivatives and errors.

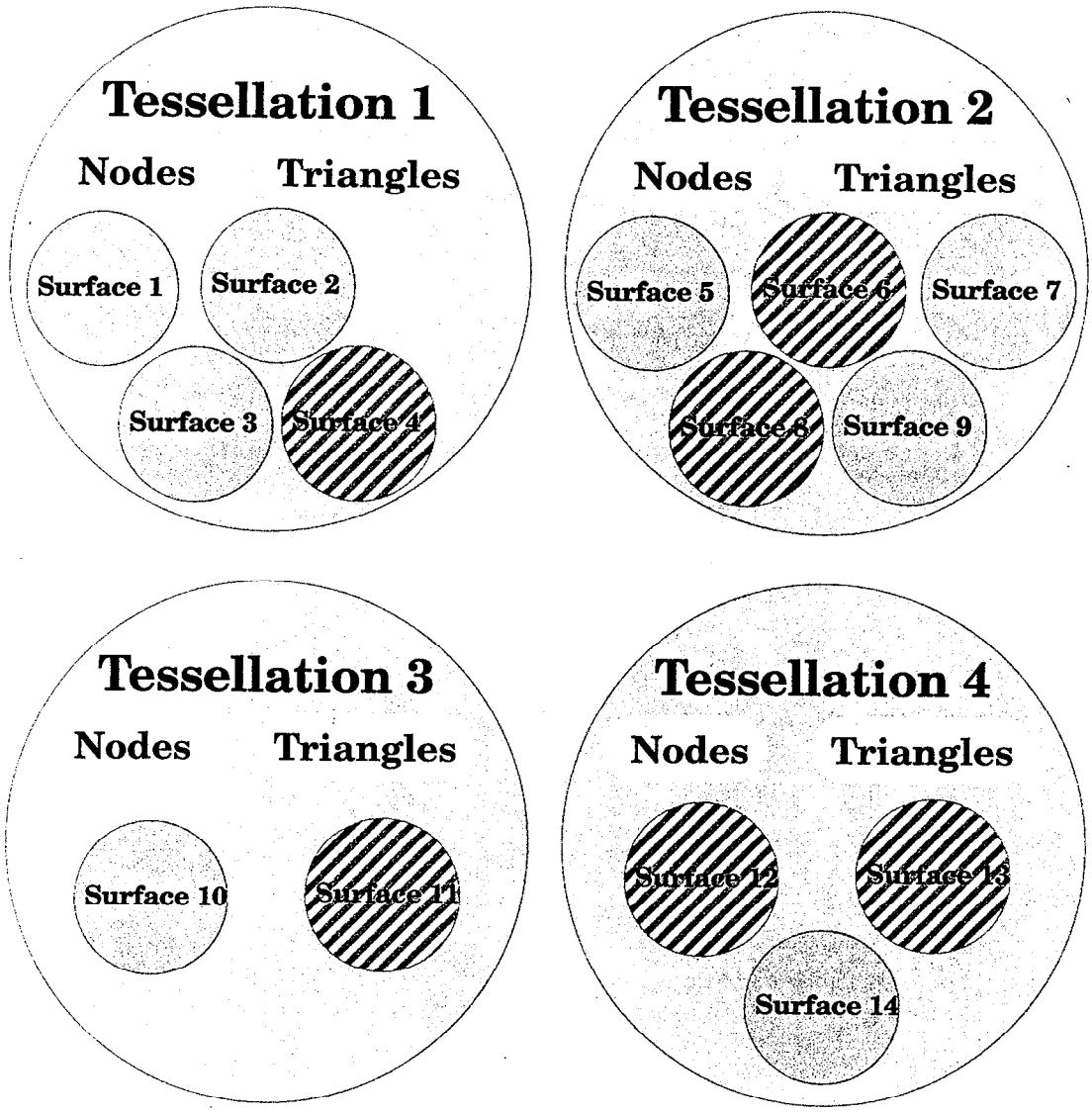


Figure 9. Surface Fetch Initialization (SFI)
Calls from the client application through libKBI can designate specific surfaces for access while restricting others. Here our imaginary KBase has been ordered to fetch surfaces 1, 2, 3, 5, 7, 9, 10, and 14 while not fetching surfaces 4, 6, 8, 11, 12, and 13.

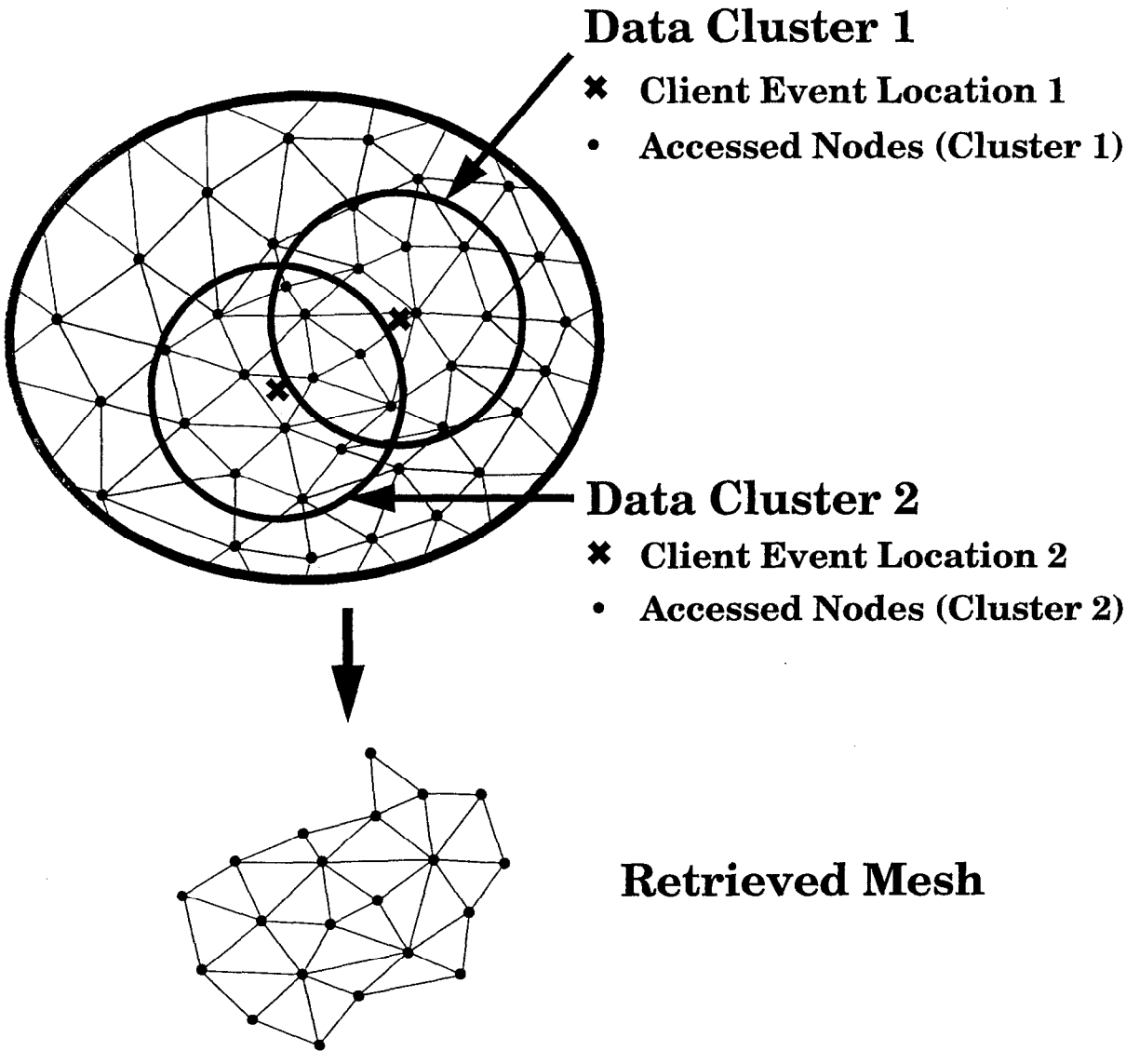


Figure 10. Data Cluster Access
Data cluster access occurs when the SDE utilizes a predefined "data cluster" (a cookie-cutter shaped like a circle in this example) to access data in the neighborhood of a guess location requested by the client application. Only nodes and triangles inside of the data cluster are accessed. Subsequent data cluster accesses (data cluster 2) return the set difference between the current data cluster and all previous data cluster accesses.

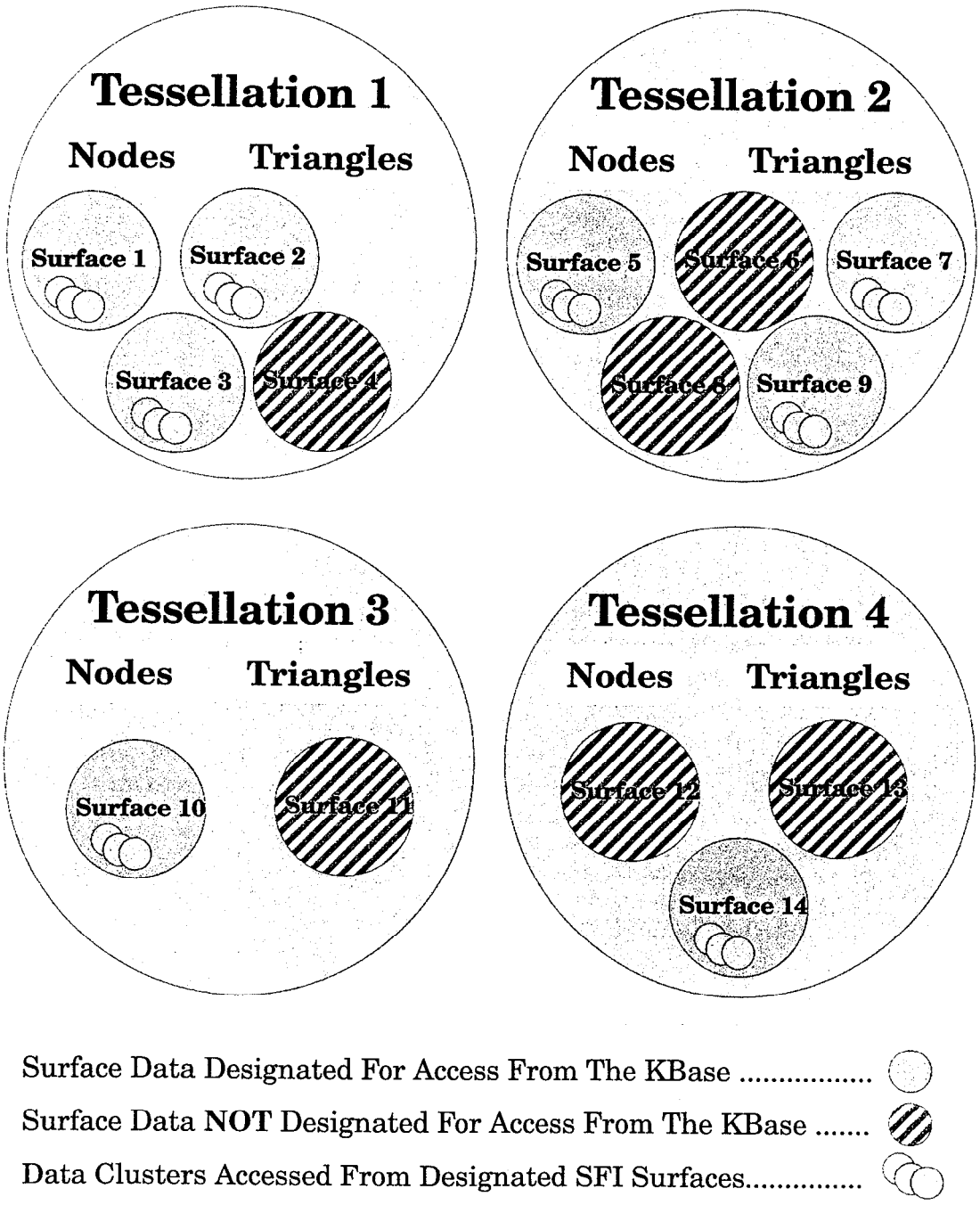


Figure 11. SFI Surface Data Cluster Access
Data cluster access affects all SFI surfaces simultaneously. In the example above the total data retrieval from the KBase following three data cluster accesses is the sum of the data contained in the union of the three data clusters as retrieved from each accessible surface.

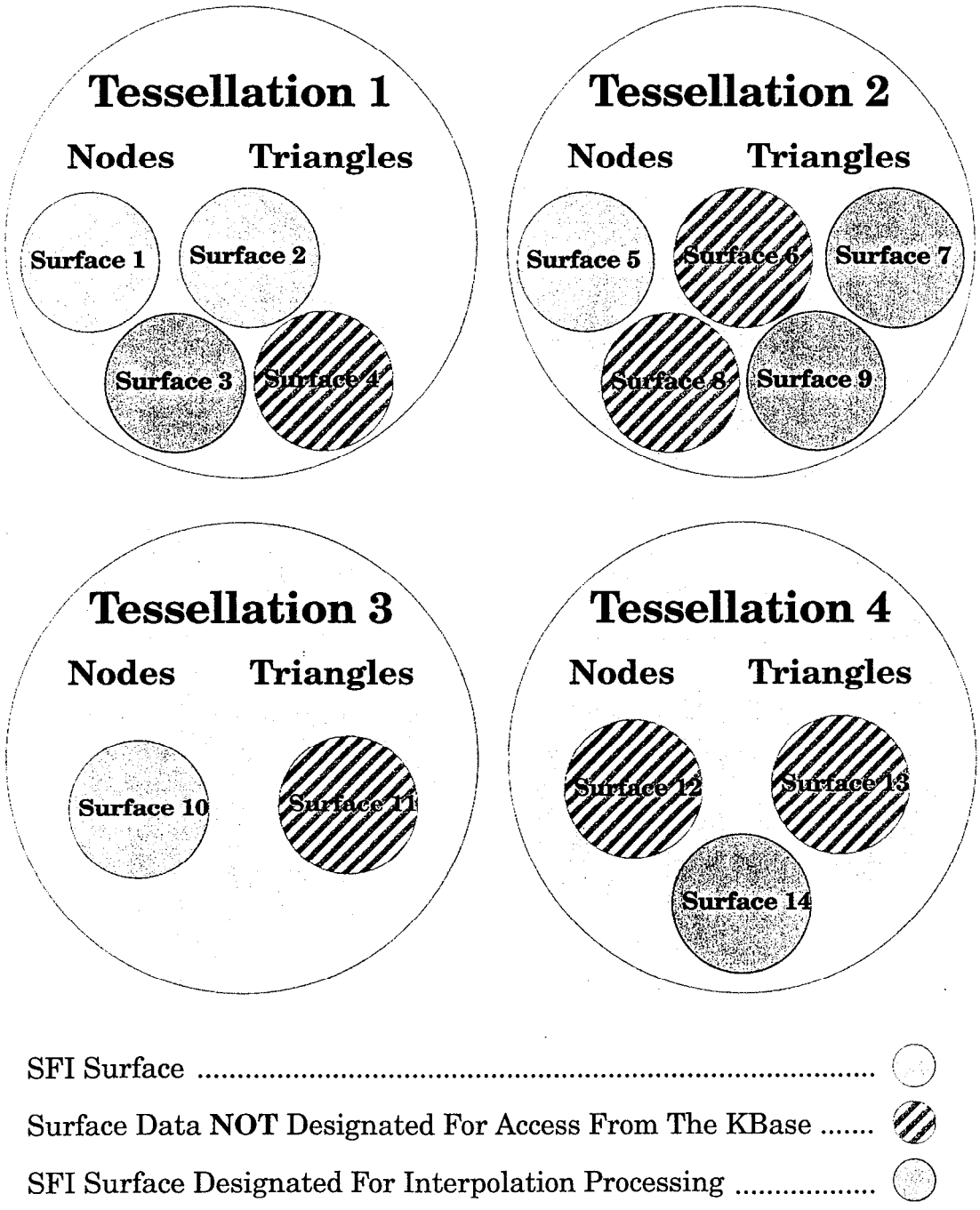


Figure 12. Surface Association
Calls from the client application through libKBI can designate specific SFI surfaces for interpolation processing. These surfaces represent those that the client application has determined as associated with the current event location analysis. In the example above surfaces 3, 7, 9, and 14 have been designated for interpolation processing.

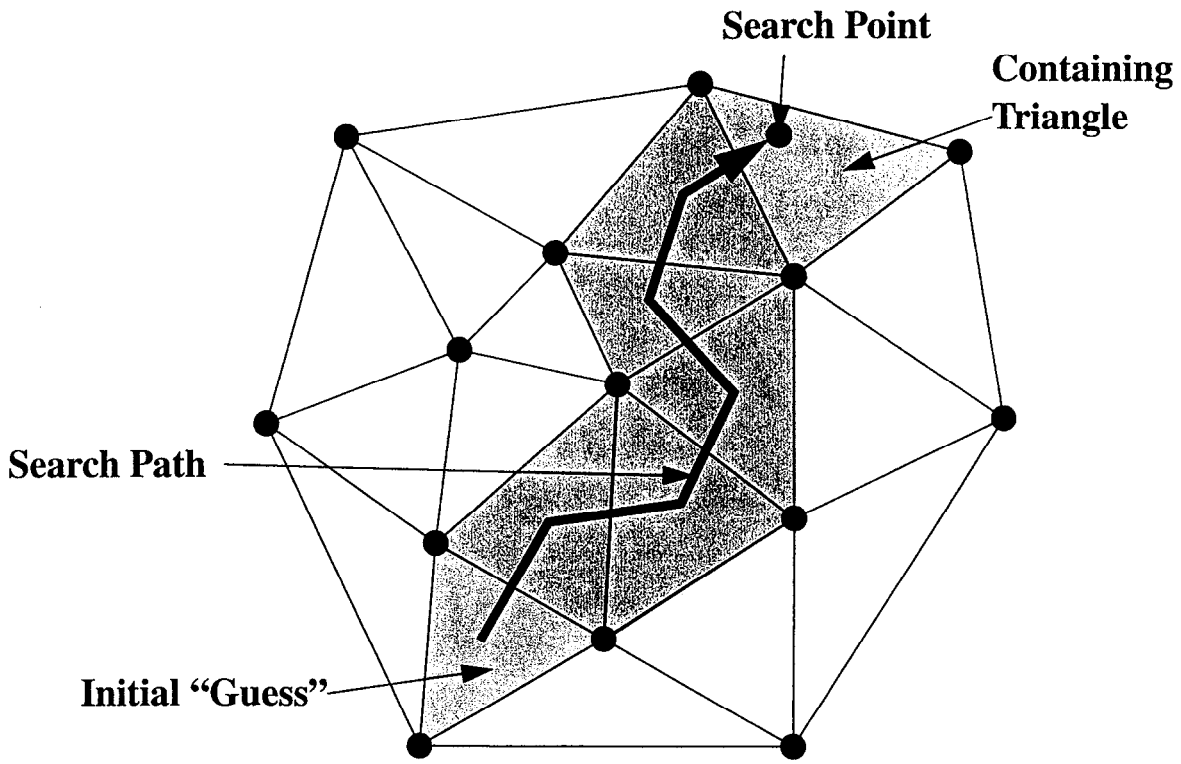


Figure 13. Walking Triangle Search
The search progresses from an initial guess to the containing triangle by continually evaluating whether the search point lies inside of or outside of the current triangle. Outside determination is accomplished by evaluating whether the search location lies to the "left" or the "right" of a counter-clockwise oriented triangle edge. If the point lies to the "left", the next edge of the triangle is subjected to the test. If the point lies to the right, however, the search shifts to the adjacent triangle that shares the edge undergoing the test and the process is repeated. The containing triangle is found when all edges test "left".

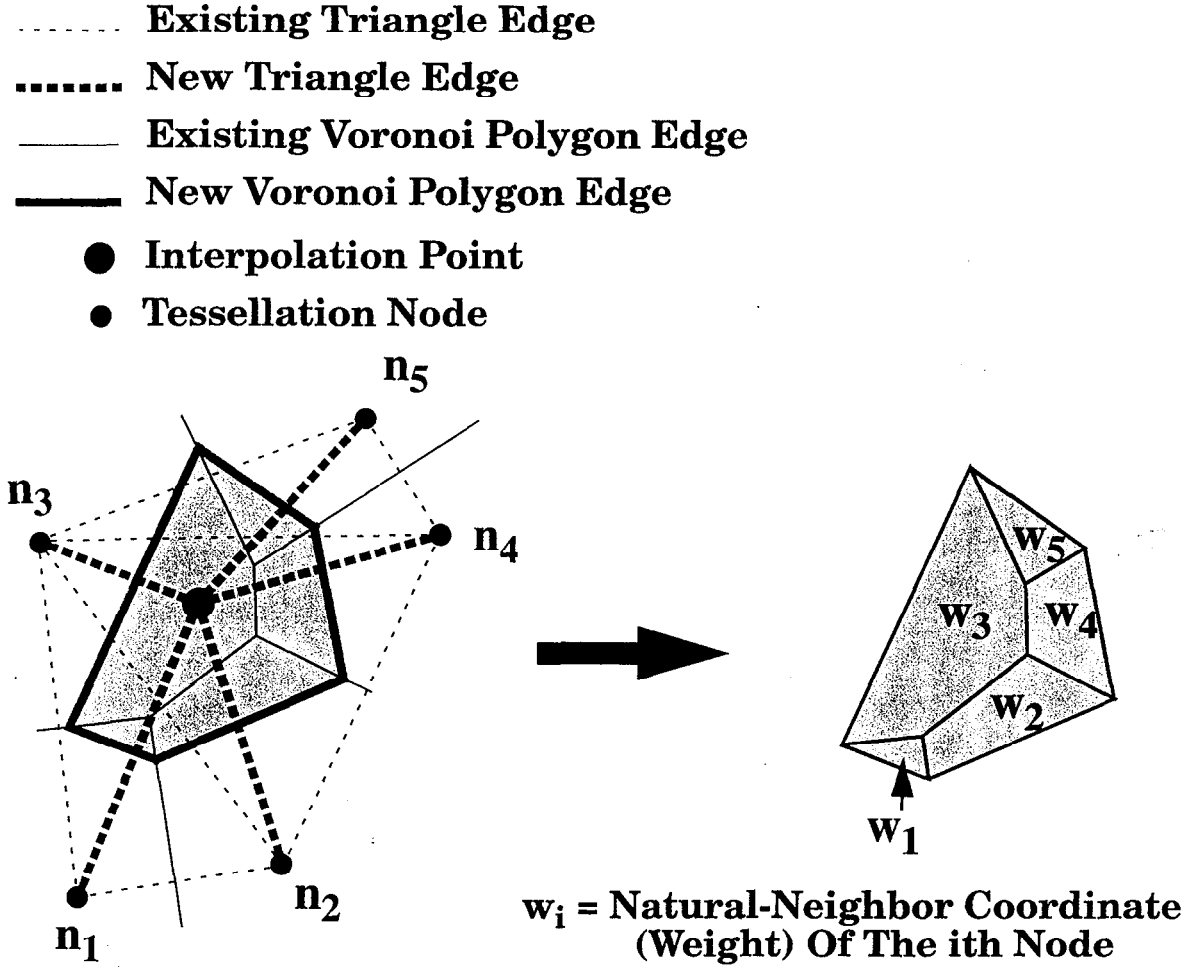


Figure 14. Natural-Neighbor weight determination
The natural-neighbor interpolation is accomplished by determining the weights that must multiply the surface values assigned to each of the neighbor nodes of the interpolation point. The weights are determined by performing a virtual insertion of the interpolation point into the existing mesh, then examining the polygonal areas of the mesh dual called the voronoi polygons. The weights are calculated as a ratio of the area of the overlap between the existing voronoi polygon of each neighbor node with the new voronoi polygon formed from the virtual insertion of the interpolation point to the entire area of the new virtual voronoi polygon.

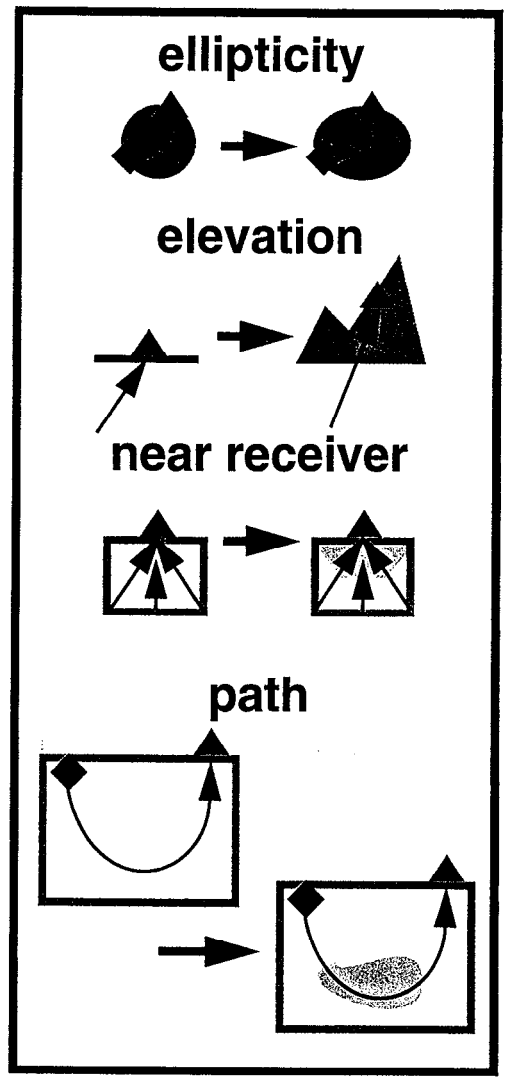


Figure 15. Base Model + Corrections approach to improving event locations
Travel times taken from a radial earth model (e.g. IASP91) are corrected for various discrepancies between that model and the real Earth: shape is elliptical, not round; stations are sited at different elevations, not all at a common height; non-radially symmetric structure occurs beneath stations affecting all paths to that station; non-radially symmetric structure occurs elsewhere along paths from sources to receivers leading to path-specific corrections.

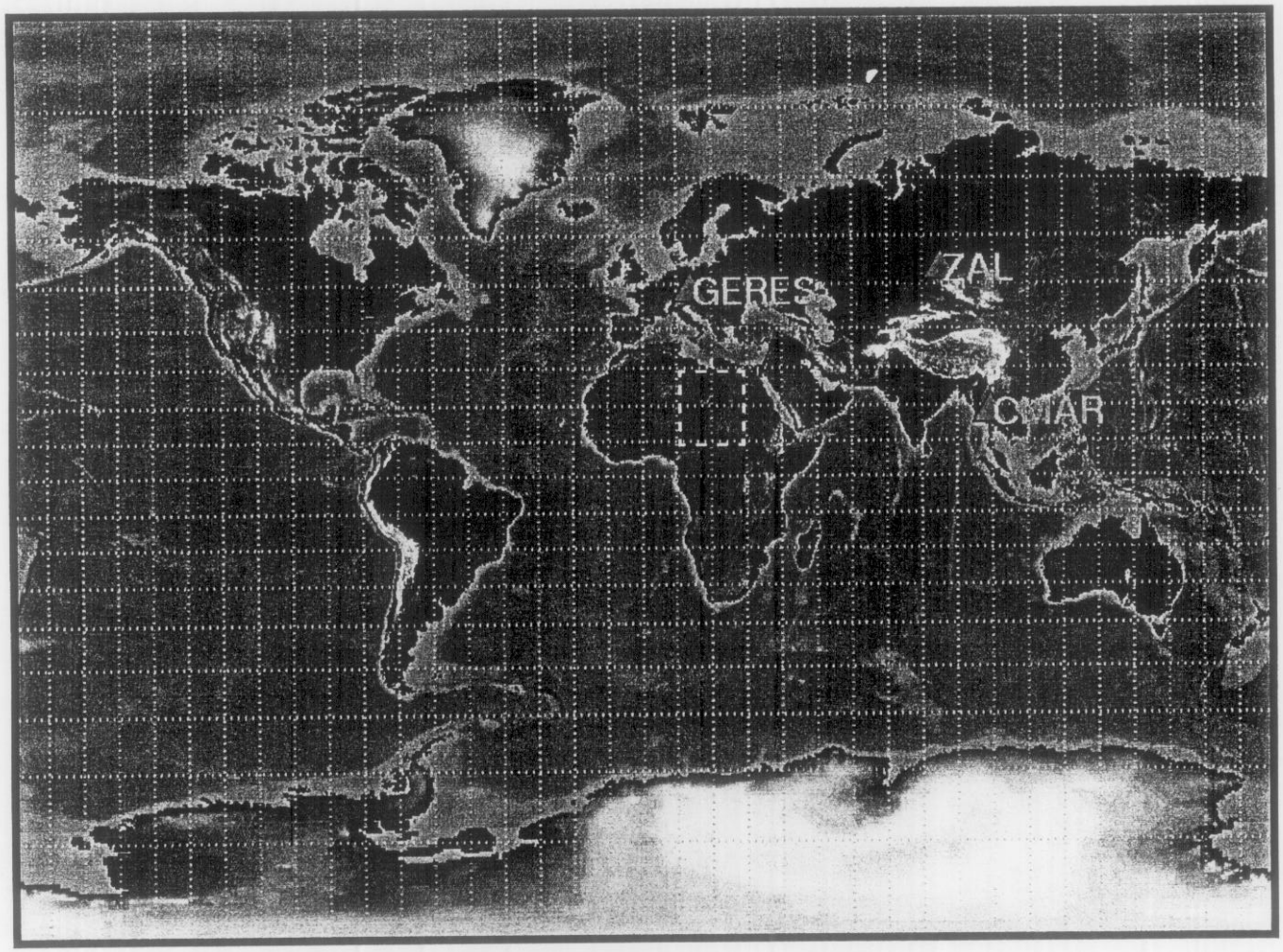


Figure 16. Stations and region used for synthetic travel time example
The dashed box indicates the region examined in this study. Three stations with poor azimuthal coverage for this area were intentionally selected to make the event locations unstable.

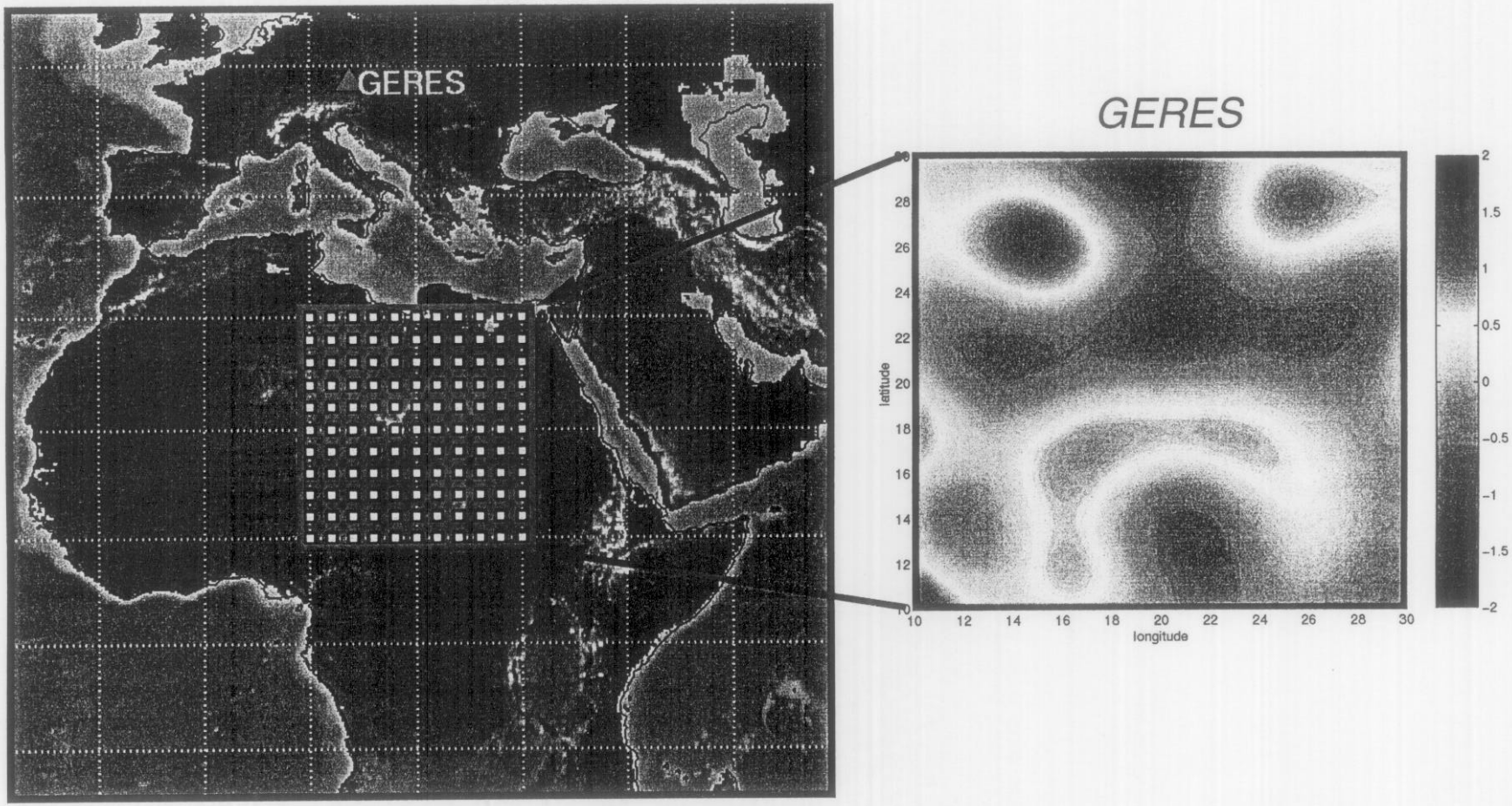


Figure 17. Synthetic test events and perturbation surface for GERES
IASP91-consistent P times (i.e. 0 residuals) for each station for the 121 test events were perturbed by adding values taken from spherical harmonic surfaces (one per station) with randomly generated order 50 coefficients of up to +/- 2 seconds.

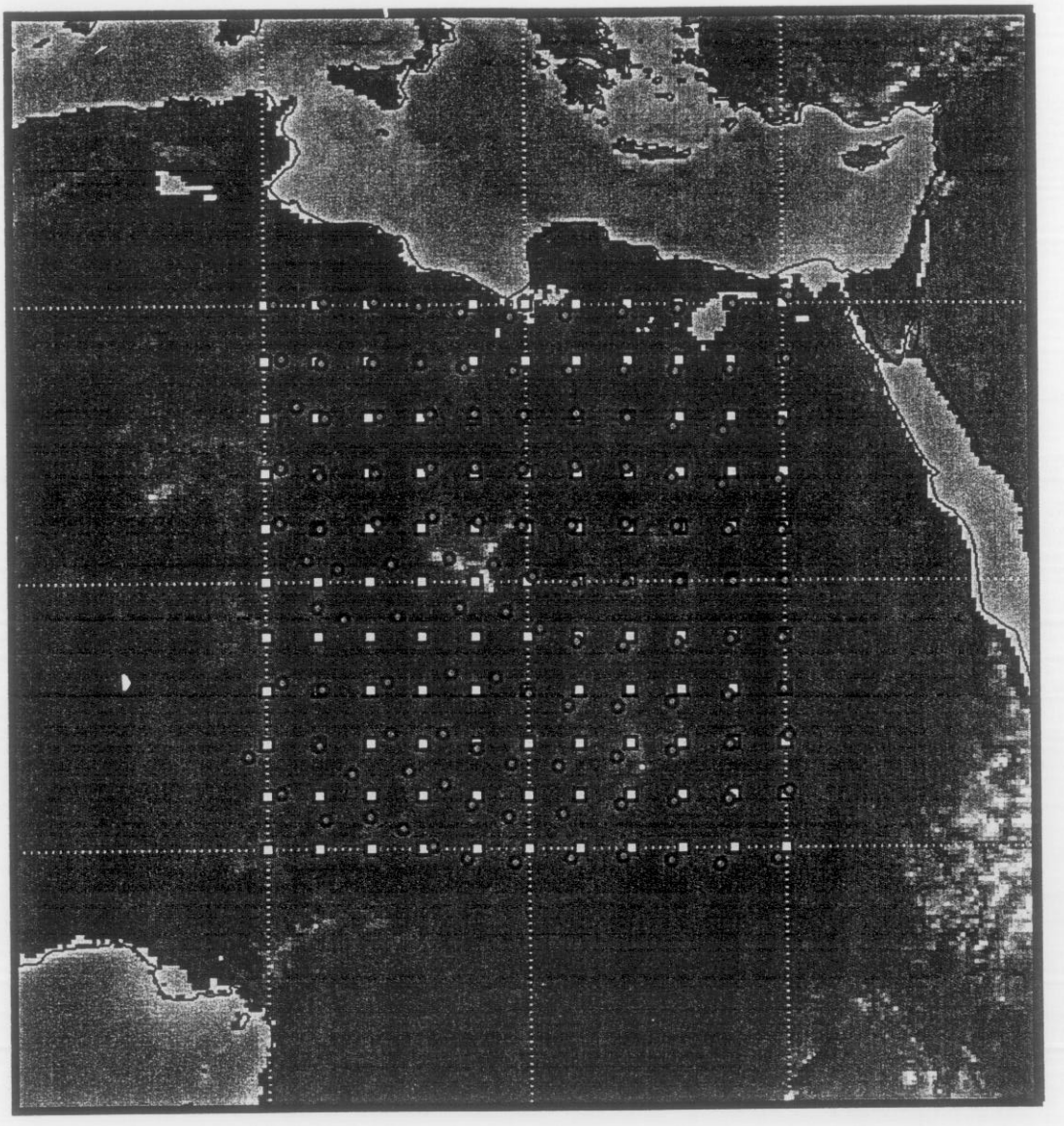


Figure 18. Re-locations of 121 perturbed events, no model error
The true event locations for the 121 test events are indicated with pink squares. The re-locations are shown with red circles. Their 95% *confidence* ellipses are also shown, but are generally too small to discern.

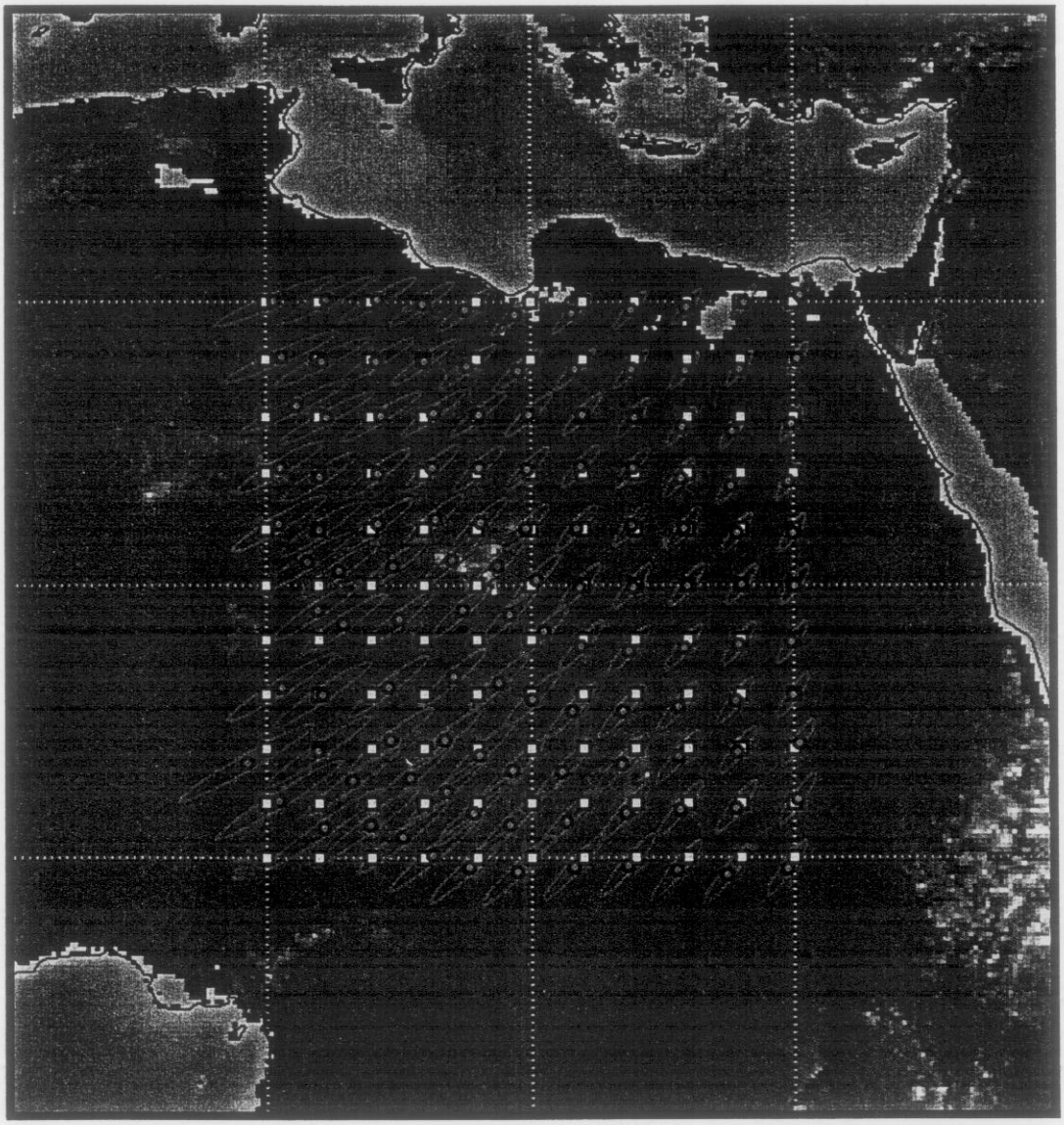


Figure 19. Re-locations of 121 perturbed events, IASP91 model error
The true event locations for the 121 test events are indicated with pink squares. The re-locations without use of the KBI are shown with red circles. The 95% *coverage* ellipses are also shown. The apriori error used accounts both for measurement and modelling error, in this case from IASP91.

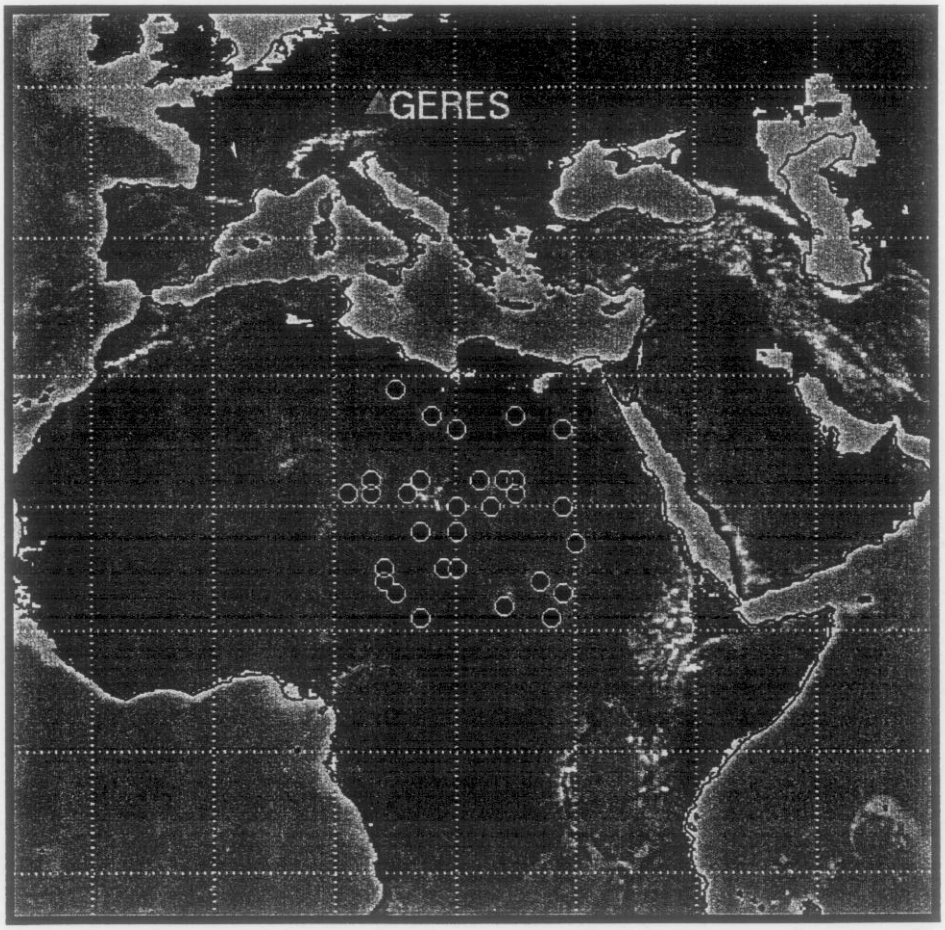


Figure 20. The 30 randomly selected ground truth points
These locations were randomly selected off a 1 degree grid spanning latitude 10-30 degrees North, and longitude 10-30 degrees East. Note that only where there is a ground truth point can we expect to recover with kriging the true surface shown in figure 17.

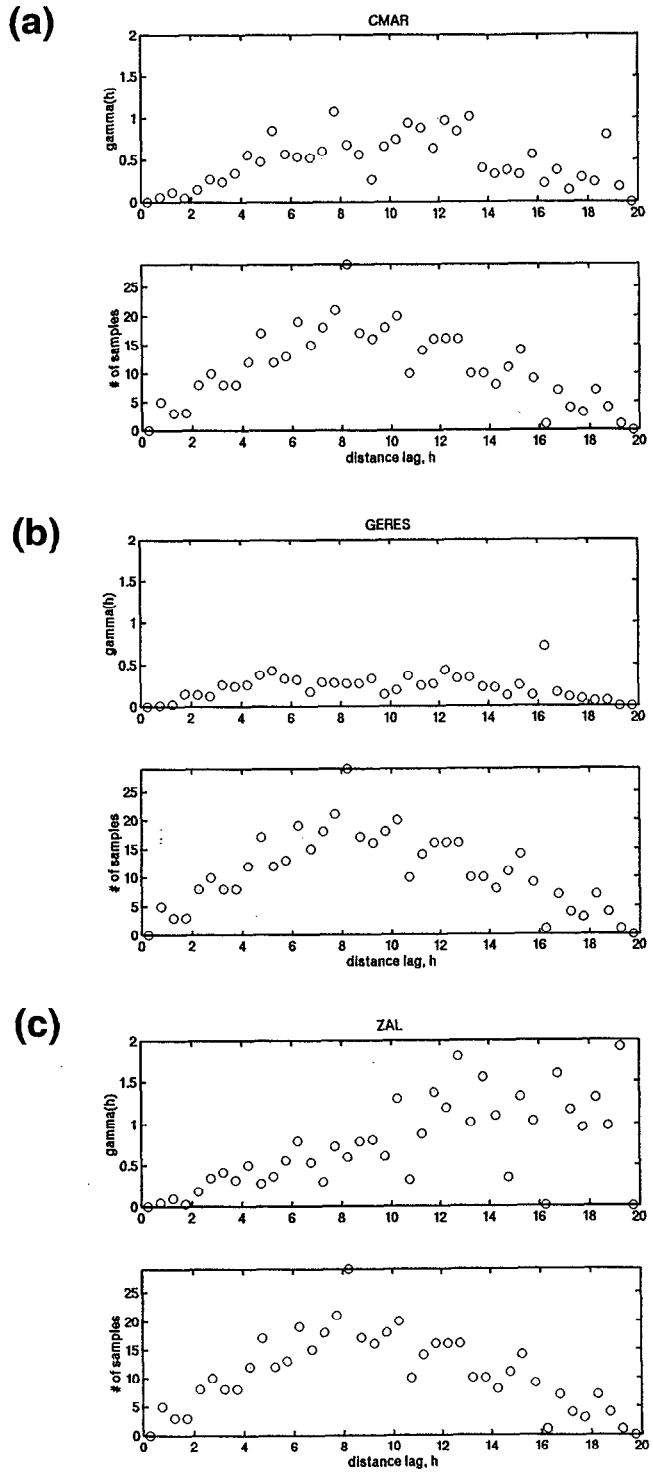


Figure 21. Variograms for CMAR, GERES, and ZAL
Observed variograms (above) and plots of number of point pairs vs. bin distance (below) for: (a) CMAR; (b) GERES; (c) ZAL.

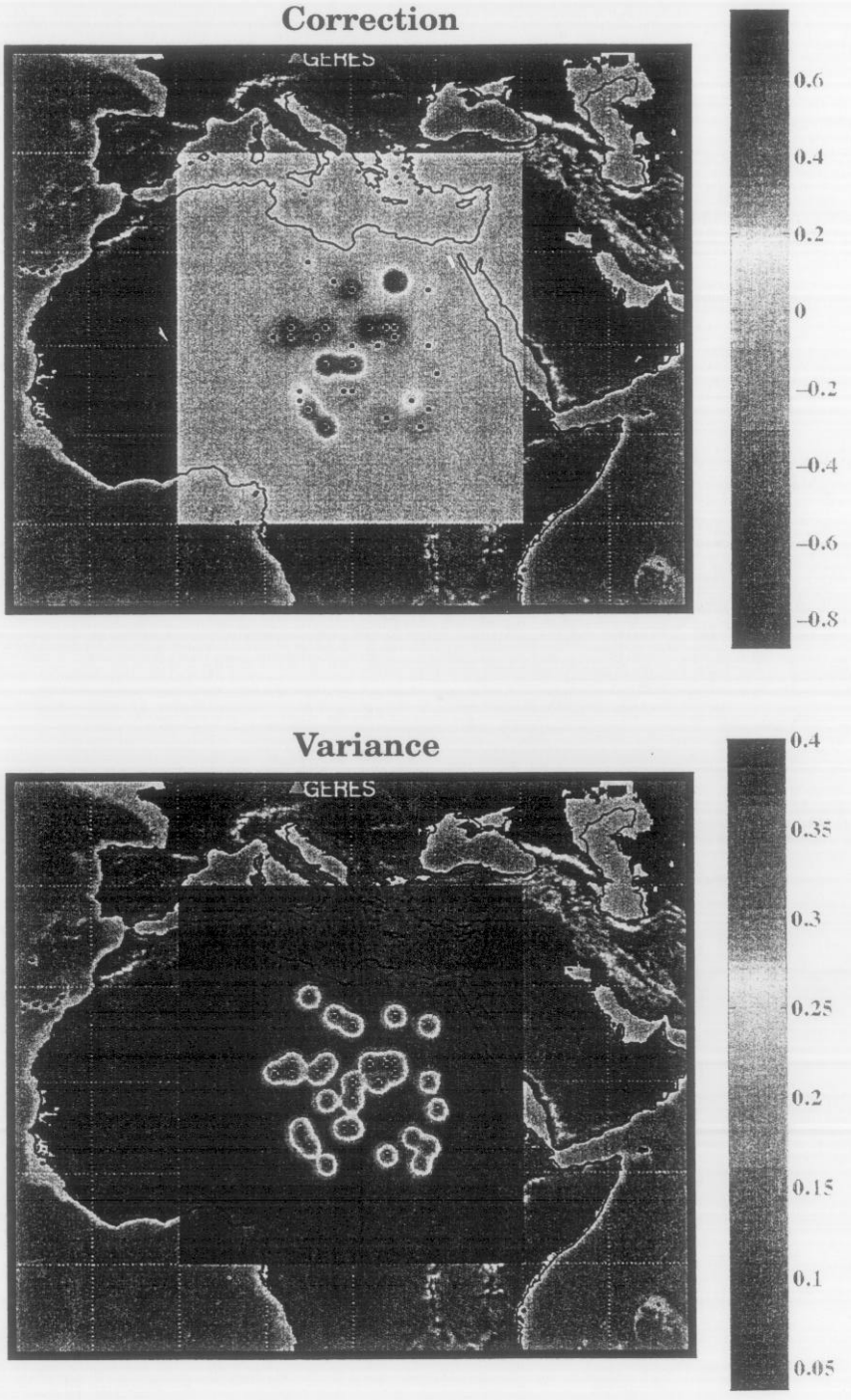


Figure 22. Kriged correction and variance surfaces for GERES
Correction (Top) and *Variance* (Bottom) surfaces for GERES derived by kriging from the 30 ground truth events. The units are *seconds* for correction, and *seconds*^{2} for variance. Note that the kriged surface matches the true surface in Figure 17 only near the ground truth points, and that the variance (error) surface indicates that our confidence in the interpolated values decreases quickly away from the ground truth points.

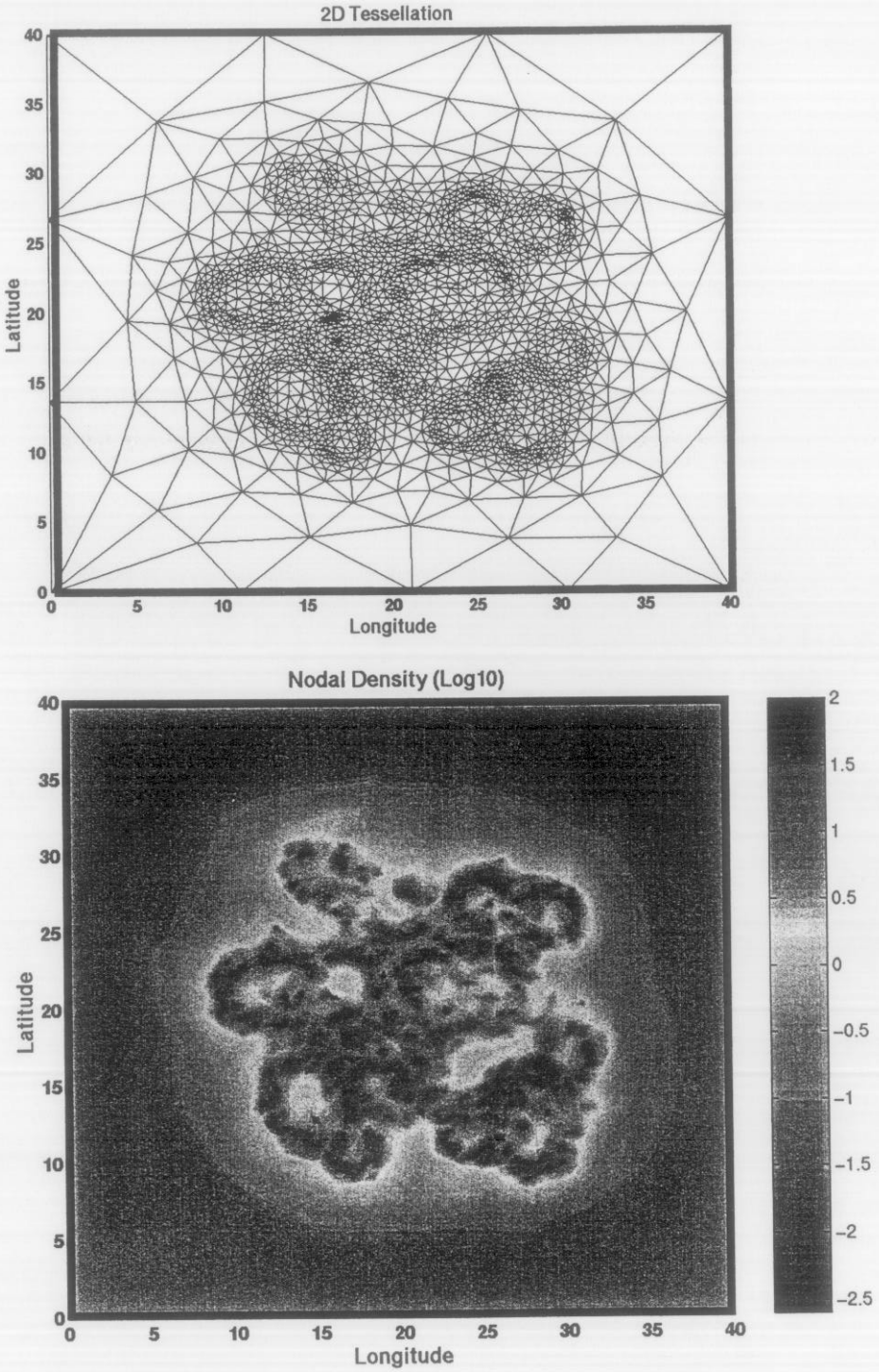


Figure 23. Final tessellation and node density for 10% error (maximum) (Top) The final tessellation developed to fit the 30 ground truth points for all three stations. The locations of the ground truth points, which were fixed, are indicated with blue circles. (Bottom) A log plot of the nodal density (units are log #nodes per square degree), showing that the nodes are densest in the regions of high gradient.

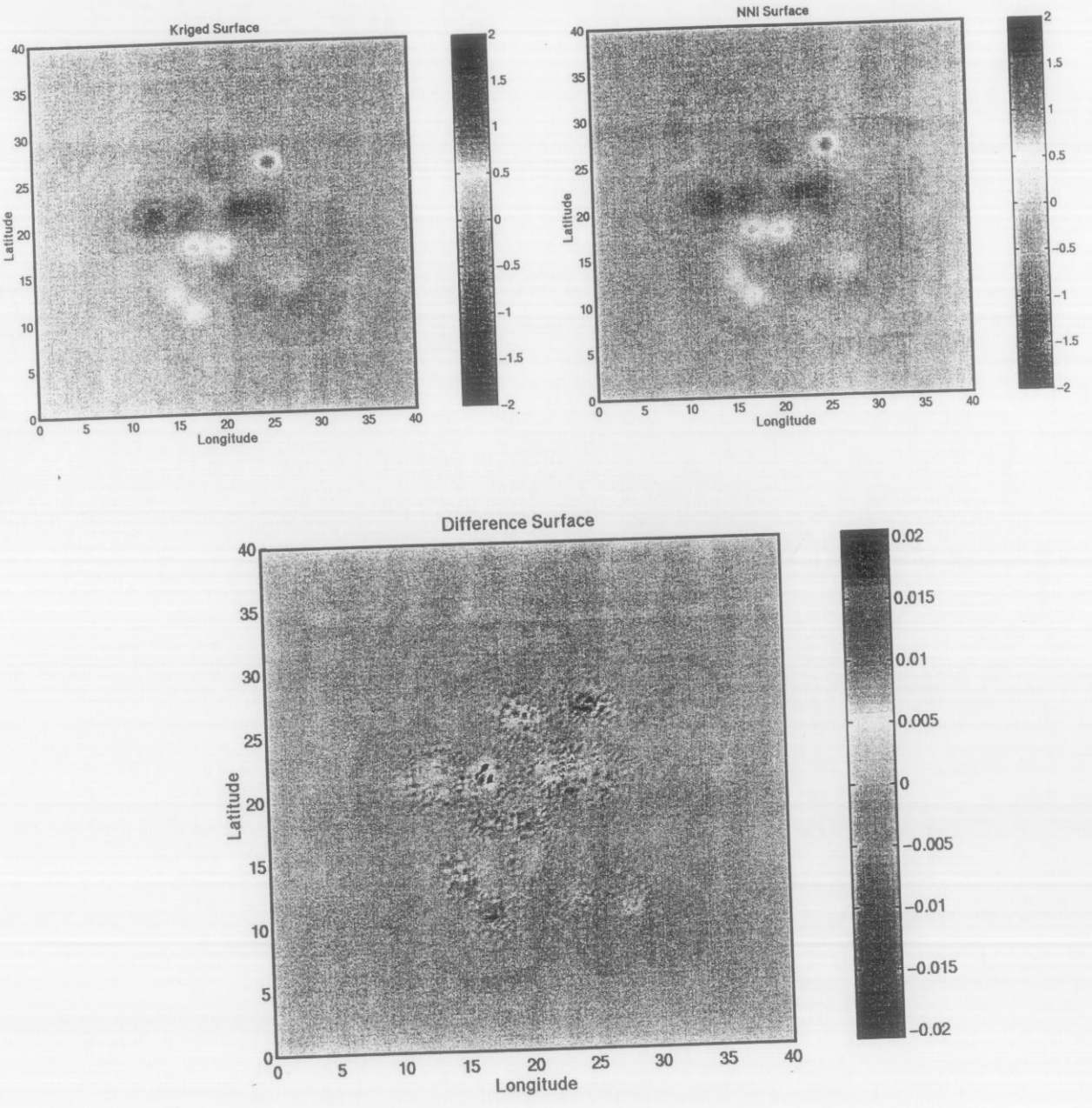


Figure 24. Comparison of Kriged and NNI surfaces for GERES
(Top left) Kriged surface from the 30 ground truth points for an 0.25 degree grid; (top right) NNI surface for the same grid using the optimal tessellation calculated for 10% error; (bottom) Difference between the two surfaces -- note the change in scale.

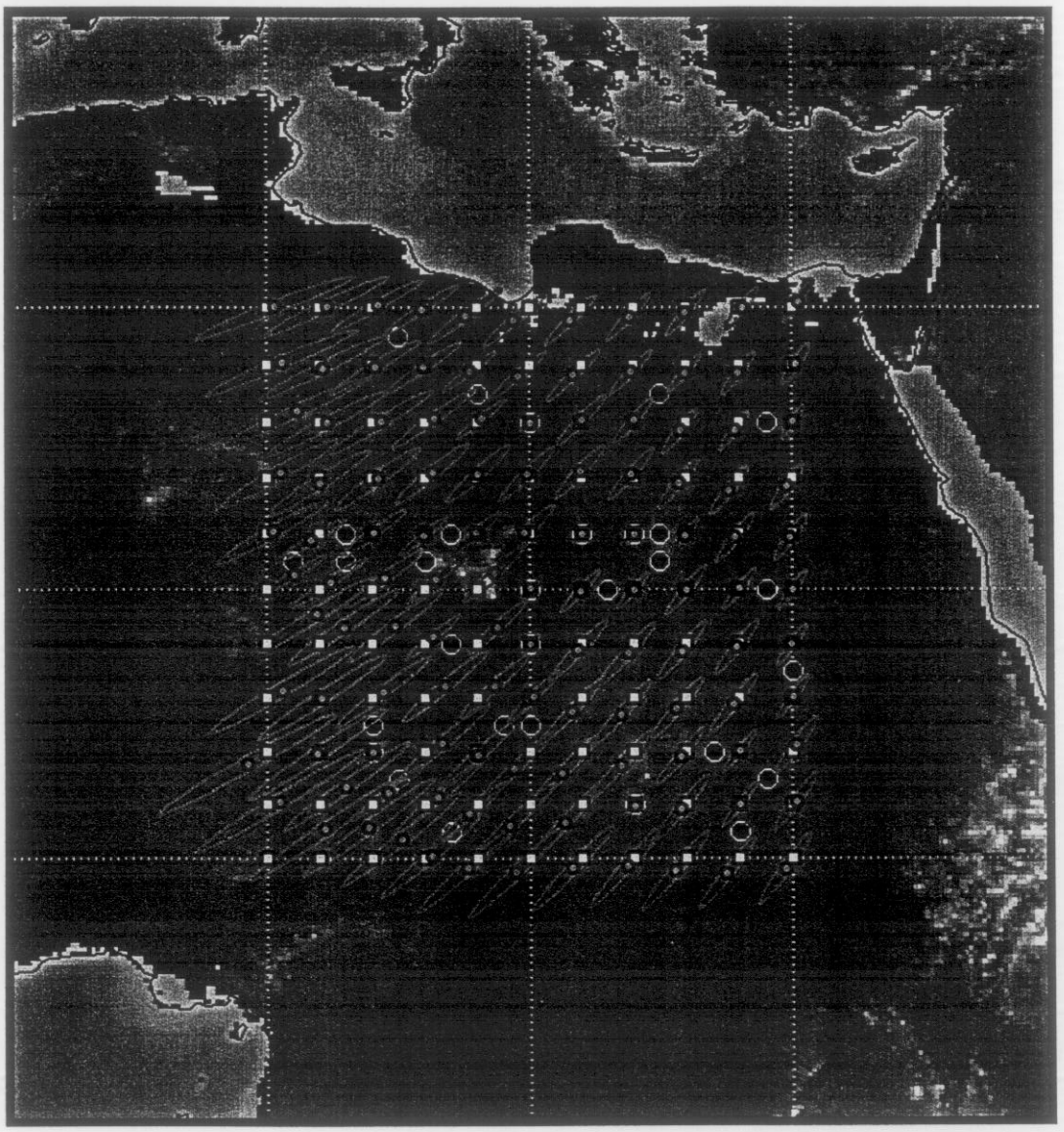


Figure 25. Relocations of 121 events, KBI used
The 30 ground truth event locations are indicated with black circles, the 121 true test event locations are indicated with pink squares, and the 121 test event re-locations with the KBI used are indicated with red circles. The 95% coverage ellipses for each relocation are also shown.

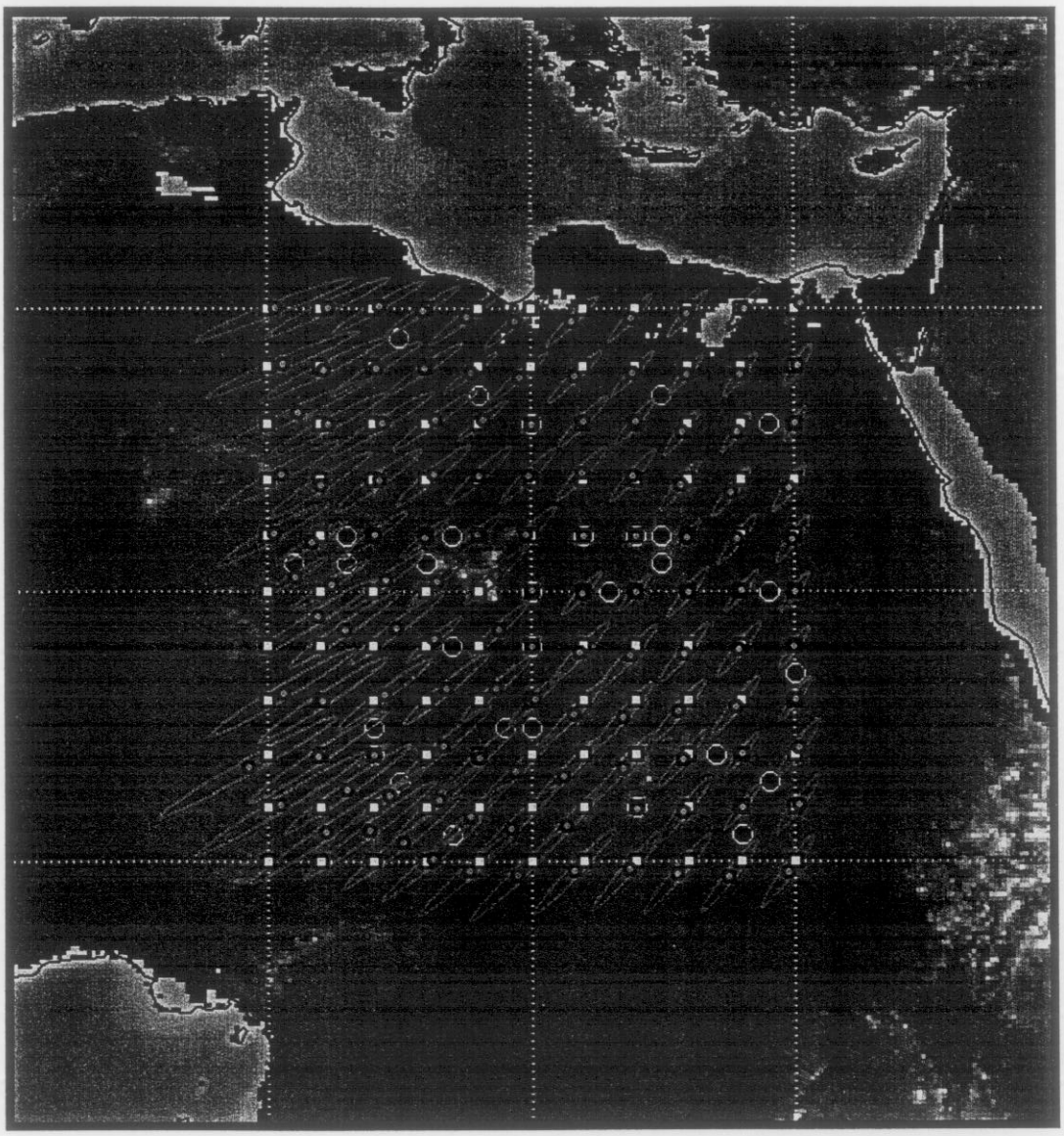


Figure 25. Relocations of 121 events, KBI used
The 30 ground truth event locations are indicated with black circles, the 121 true test event locations are indicated with pink squares, and the 121 test event re-locations with the KBI used are indicated with red circles. The 95% coverage ellipses for each relocation are also shown.

Appendix A: Modified Bayesian Kriging

A-1 Introduction

This appendix describes the derivation of a modified kriging technique to be used to generate interpolated data and error estimates for data sets to be included in the DOE Knowledge Base. The technique was developed by personnel from Sandia National Laboratories, Lawrence Livermore National Laboratories, Pacific Northwest National Laboratories, and the Air Force Technical Applications Center (AFTAC). It was developed to satisfy several major requirements of the Knowledge Base data sets that existing kriging methods do not satisfy very well, or at all. These requirements include:

- The capability to produce smooth (continuous first and second derivatives) value and variance surfaces to meet the needs of the current seismic event location algorithm (EvLoc).
- The capability to model data variance independently for each point or a group of points in such a way that the interpolation surface follows the "good" points more closely than the "bad" points. This provides a mechanism to combine all available data (e.g. earthquakes and nuclear tests)
- The capability to vary the range of influence of each data point or group of data points, independent of the quality of the data point
- The capability to make both the interpolated value surface and variance surface blend to background values beyond the range of influence of any observations, so that continuity from regional to teleseismic domains can be assured.
- The capability to provide smooth and continuous residual-to-zero and error-to-background transitions across user specified boundaries (e.g. faults or tectonic provinces). This allows the modeler to control the influence of subsets of data and yet still assures continuity across the boundaries.

Using any of several types of kriging to form a basis for gradient-modified NNI, we can satisfy the first requirement. To meet the other four requirements we choose to work specifically with Simple Kriging (kriging with known mean) and make two major modifications: the incorporation of measurement (i.e. uncorrelated) error by deriving the Simple Kriging equations using covariance instead of variogram functions, and the introduction of blending or damping functions through Bayesian methodology. For clarity we will discuss each of these modifications separately, ultimately building up to the complete proposed system.

A-2 Derivation

A-2.1 Simple Kriging Equations with Uncorrelated Error

We begin with a derivation of the Simple Kriging equations with uncorrelated error. We choose to base our derivation on Simple Kriging rather than Ordinary Kriging because we assume that the mean is known.

Consider a random function Z of spatial coordinate vector $\vec{X}$ from which any systematic trend has been removed. Then Z can be represented as some stationary spatially correlated component (Θ) plus a non-correlated component (ξ), i.e. noise):

$$Z(\vec{X}) = \Theta(\vec{X}) + \xi(\vec{X}) \qquad \text{(EQ A-1)}$$

We will assume that both Θ and ξ are zero mean, which implies that the data has been demeaned. Suppose that we know the value of Z at several points and that we wish to predict the value of Z at an arbitrary point p. This is a standard interpolation problem, and one solution is to let our predicted value $Z^*(\vec{X}_p)$ be a linear weighted sum of the values of Z at the known points. For Simple Kriging (i.e. kriging with known mean), the interpolation equation is

$$Z^*(\vec{X}_p) = m + \sum_{i=1}^{n} w_i[Z(\vec{X}_i) - m] \qquad \text{(EQ A-2)}$$

where m is the mean of Z. For our derivation, Z is assumed to be zero mean so we can re-write (A-2) as

$$Z_p^* = \sum_{i=1}^{n} w_i Z_i \qquad \text{(EQ A-3)}$$

where we have written Z_i for $Z(\vec{X}_i)$ for compactness and will continue to do so henceforth.

Any number of interpolation methods can be used to find the "best" weights w_i. For kriging we seek to find the weights that minimize the variance of the prediction error at $\vec{X}_p$. The prediction error variance is

$$\sigma_{ep}^2 = EZ_p - \sum_{i=1}^{n} w_i Z_i^2 \qquad \text{(EQ A-4)}$$

where E indicates the expectation operator, and we have assumed that the prediction error has zero mean. The minimization of this equation is the basis for kriging.

At this point in most standard derivations, the variance equation is rearranged into terms involving differences between the points Z_i, Z_j, and Z_p, so that these differences can be replaced with theoretical variogram functions. We choose not to do this because to use a variogram, we must assume that all of the points have the same variance, and this would violate the second of our requirements. Instead we will work with covariances which will allow us to individually specify the variance for each point.

We begin by writing out the square on the right hand side of equation (A-3)

$$\sigma_{ep}^2 = E\left\{ Z_P^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j Z_i Z_j - 2\sum_{i=1}^{n} w_i Z_P Z_i \right\} \qquad \text{(EQ A-5)}$$

We will re-write this equation, making appropriate substitutions for each term on the right hand side. For the interpolation point p, we assume that $Z_P = \Theta_P$ only, i.e. $\xi_P = 0$, so the predicted value will have no uncorrelated component. Also, we will use the zero-mean stipulation above to make the substitutions

$$E(\Theta_i\Theta_j) = cov(\Theta_i, \Theta_j) \qquad \text{(EQ A-6)}$$

$$E(\xi_i\xi_j) = cov(\xi_i, \xi_j) \qquad \text{(EQ A-7)}$$

Using Equation (A-6), we can re-write the expected value of the square of the value at point p as

$$E\{Z_P^2\} = cov(\Theta_p, \Theta_p) = \sigma_{\Theta_P}^2 \qquad \text{(EQ A-8)}$$

Using (A-6) and (A-7) the expected value of the product of the values at i and j can be re-written as

$$E\{Z_iZ_j\} = E\{\Theta_i\Theta_j + \xi_i\xi_j + \Theta_i\xi_j + \Theta_j\xi_i\} = cov(\Theta_i, \Theta_j) + cov(\xi_i, \xi_j) \qquad \text{(EQ A-9)}$$

$$= cov(\Theta_i, \Theta_j) + cov(\xi_i, \xi_j)$$

$$= cov(\Theta_i, \Theta_j) + \delta_{ij}\sigma_{\xi_i}\sigma_{\xi_j}$$

where δ_{ij} is the del operator (i.e. = 1 when i=j, = 0 otherwise), and the cross terms involving both Θ and ξ have dropped out because the ξ_i's do not correlate with anything but themselves.

Similarly, the expected value of the product of the values at i and p is

$$E\{Z_PZ_i\} = E\{\Theta_P\Theta_i + \Theta_P\xi_i\} \qquad \text{(EQ A-10)}$$

$$= cov(\Theta_P, \Theta_i)$$

Substituting (A-8), (A-9), and (A-10) into (A-5) we get

$$\sigma_{ep}^2 = \sigma_{\Theta_P}^2 + \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j (cov(\Theta_i, \Theta_j) + \delta_{ij}\sigma_{\xi_i}\sigma_{\xi_j}) - 2\sum_{i=1}^{n} w_i (cov(\Theta_P, \Theta_i)) \qquad \text{(EQ A-11)}$$

We seek the particular set of weights which will *minimize* σ_{ep}^2, so we will differentiate (A-11) with respect to w_i and set the result equal to zero. This is the point at which an unbiased constraint (i.e. $\sum_{i=1} w_i = 1$) can be introduced if desired, using the standard Lagrange parameter.

For our derivation, we do not choose to do so.

Now we take the derivative of (A-11) and set the result equal to zero, to get the kriging equations:

$$\sum_{j=1}^{n} w_j (cov(\Theta_i, \Theta_j) + \delta_{ij}\sigma_{\xi_i}\sigma_{\xi_j}) = cov(\Theta_P, \Theta_i) \qquad \text{(EQ A-12)}$$

Substituting this result into (A-11) we find the equation for the variance

$$\sigma_{ep}^2 = \sigma_{\Theta_P}^2 - \sum_{i=1}^{n} w_i cov(\Theta_P, \Theta_i) \qquad \text{(EQ A-13)}$$

Using the relationship $cov(X, Y) = \rho_{XY}\sigma_X\sigma_Y$, where ρ is the correlation coefficient, we can rewrite these equations as

$$\sum_{j=1}^{n} w_j (\rho_{\Theta_i\Theta_j}\sigma_{\Theta_i}\sigma_{\Theta_j} + \delta_{ij}\sigma_{\xi_i}\sigma_{\xi_j}) = \rho_{\Theta_P\Theta_i}\sigma_{\Theta_P}\sigma_{\Theta_i} \qquad \text{(EQ A-14)}$$

and

$$\sigma_{ep}^2 = \sigma_{\Theta_P}^2 - \sum_{i=1}^{n} w_i \rho_{\Theta_P\Theta_i}\sigma_{\Theta_P}\sigma_{\Theta_i} \qquad \text{(EQ A-15)}$$

To use these equations to find estimates for Z_P and σ_P^2, we must supply values for $\rho_{\Theta_i\Theta_j}$, σ_{Θ_i} and σ_{ξ_i}. σ_{ξ_i} is just the uncorrelated standard error associated with each point i, i.e. the measurement error. The presence of this term allows us to satisfy the second of our requirements. The other two terms -- the correlation coefficient func-

tion between points i and j, and the variance of the values at those points -- can be taken from a variogram calculated for the data set to be kriged (Fig. A-1).

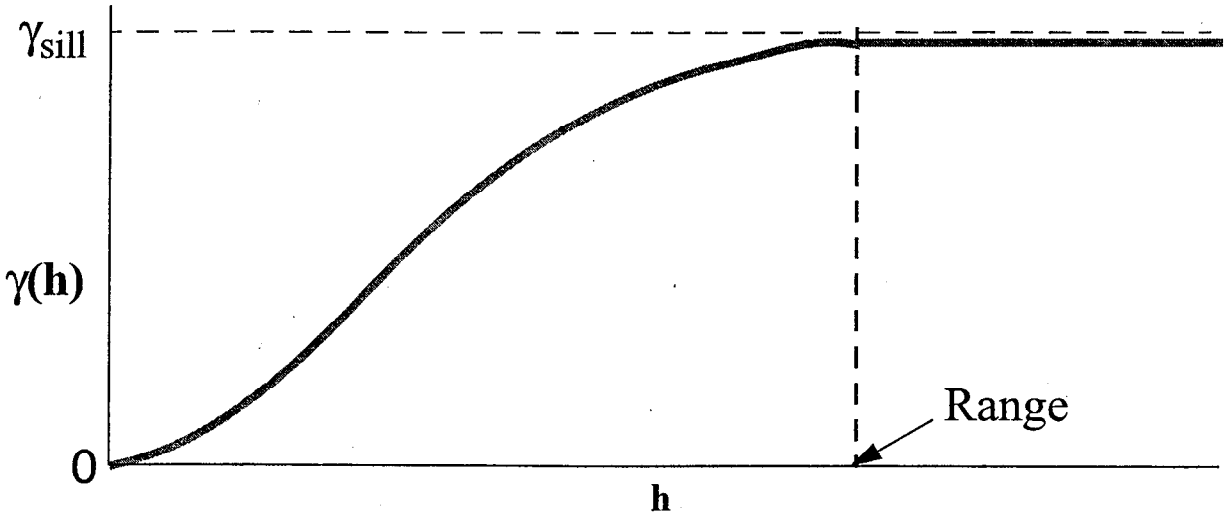


Figure A-1. Typical Variogram (with zero nugget)

Essentially, to get the correlation coefficient function we ignore the nugget, normalize and flip the variogram function. The correlation coefficient function then scales from 1 to 0 over the range of the variogram. Thus, the value for $\rho_{\Theta_i\Theta_j}$ depends only on the distance between Z_i and Z_j. The σ_{Θ_i} terms are all the same (i.e. independent of i) and can be taken from the sill value of the variogram (i.e. the background variance due to the correlated processes).

A-2.2 Correlated Variance vs. Uncorrelated Variance

Before introducing the Bayesian modifications, we first want to clarify the difference between the variances of the two different types of processes referred to in Equation (A-1): variance due to a correlated random process and variance due to an uncorrelated random process. To understand the difference, let us consider a few simple examples.

First, suppose that we have a model that can exactly predict the travel time between any two points in the Earth, and assume that we have a seismic event whose location is known perfectly, but for which the arrival observed at a given station is emergent (i.e. the exact timing of the first arrival is ambiguous). The residual time (relative to our model) measured for this arrival is non-zero due to the uncorrelated random component which we refer to as *measurement error.* The measurement error will have no spatial correlation from point to point. For example, it is entirely possible that another event co-located with the first event might have a very impulsive arrival at the same station leading to a trivially small measurement error: the measurement errors associated with the two points are completely uncorrelated. Taking a large set of such event points observed at the station, we can establish the variance and the covariance due to the measurement error. We will find that the covariance between different points is zero: i.e. the random process has no spatial correlation. If we instead consider a case where the arrival is impulsive (i.e. measurement error free) but the assumed location is incorrect, the situation is essentially the same (unless the mislocation is systematic and can be correlated from event to event). The net effect is again to introduce a random uncorrelated process into the travel time residual, in this case due to the mismatch between the true location and the assumed location.

Now let us consider an example of a correlated random process. Suppose that our model does not exactly predict travel times, e.g. it assumes a radial Earth, but that in fact the Earth has lateral heterogeneity (the true situation!). Now assume that we have a set of events for which the locations are known exactly (e.g. nuclear explosions identified by satellite photos). If we calculate the travel time residuals for the events they will be non-zero and again we can measure the variance and the covariance. In this case, the covariance will depend on distance (and location) indicating that the "random" process accounting for the non-zero values is spatially correlated. This is the variance due to the correlated random process (i.e the lateral heterogeneity of the Earth).

In general, of course, both types of processes are present. Even if we use a more detailed model than a radial model such as IASPEI '91, some component of the true structure will go unmodeled and this will lead to the presence of a spatially correlated random process. Conversely, many of the event locations will not be known perfectly and even for those that are, travel time picks will have some amount of measurement error, so an uncorrelated random process will also be present.

A-2.3 Bayesian Modification of the Uncorrelated Error: Blending Functions

With our derivation of the equations for Simple Kriging with zero mean in terms of variances and correlation coefficients, we now have a means to calculate the interpolated residual at any arbitrary point such that the second of our requirements is met. This technique will work well when interpolating between points but will fail to meet our third requirement away from points. Further, at present, we have no means to control the range of influence of each point, as is needed to meet the fourth requirement. We can meet both requirements by modifying the kriging equations using a Bayesian approach.

Following the lead of Omre (1984), we assume that covariance can be written as

$$cov_{Z,M}(Z_a, Z_b) = cov_{Z|M}(Z_a, Z_b) + cov_M(Z_a, Z_b) \qquad \text{(EQ A-16)}$$

where M refers to the a priori model, and Z | M is "Z conditional on M". This equation says that the covariance for the *joint* distribution of the blended random process Z and the model, M, is equal to the covariance for the *conditional* distribution of Z dependent on M plus the covariance for the *model* M. The purpose of introducing Bayesian methodology is to form a smooth transition between modeled and empirical behaviors; we want the previously derived kriging equations to apply when data sampling is good, but we also want to introduce a modeled behavior when data sampling is not good. If we can find expressions for $cov_{Z|M}(Z_a, Z_b)$ and $cov_M(Z_a, Z_b)$, we can combine these to get $cov_{Z,M}(Z_a, Z_b)$ and substitute this expression into (A-14) and (A-15) to get Bayesian kriging equations.

To figure out what the expressions must be, let us state the desired behavior. If the interpolation point is close to a data point, then the model should have no effect and equations (A-14) and (A-15) should apply, that is the covariances should be as expressed in our derivation. However, as the interpolation point moves far away from data points, the covariances from our derivation should fade to zero and the covariances should come from the model instead. We will provide expressions which accomplish this below, but first we must introduce the concept of a normalized blending function for each data point i which smoothly transitions from 1 to 0 over some range r as a function of increasing distance from the data point i to the interpolation point p. We denote this function as $B_i(|\vec{X}_i - \vec{X}_p|, r_i)$ for the i^{th} point evaluated at dis-

tance $|\hat{X}_i - \hat{X}_p|$ with a range of r_i. For compactness, we will refer to this function as B_{ip}. Figure (A-2) below illustrates a typical normalized blending function.

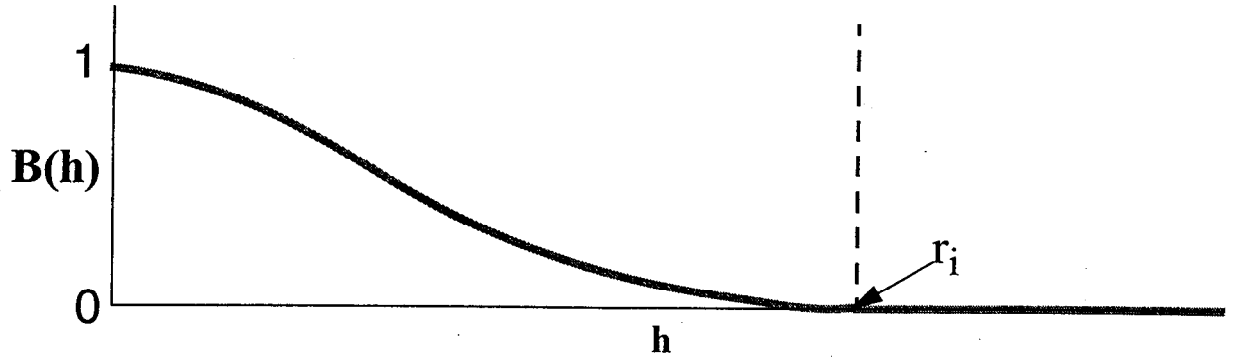


Figure A-2. Normalized blending function
The blending function is equal to 1 at h = 0, and transitions to zero at h = r_i, the range of the i^{th} data point.

Using this function, we write expressions for conditional and a priori covariances as

$$cov_{Z|M}(\Theta_i, \Theta_j) = B_{ip}B_{jp}[cov(\Theta_i, \Theta_j)] \quad \text{(EQ A-17)}$$

$$cov_M(\Theta_i, \Theta_j) = \delta_{ij}cov(\Theta_i, \Theta_j)[1 - B_{ip}B_{jp}] \quad \text{(EQ A-18)}$$

$$cov_{Z|M}(\xi_i, \xi_j) = B_{ip}B_{jp}[cov(\xi_i, \xi_j)] \quad \text{(EQ A-19)}$$

$$cov_M(\xi_i, \xi_j) = \delta_{ij}cov(\xi_i, \xi_j)[1 - B_{ip}B_{jp}] \quad \text{(EQ A-20)}$$

The a priori model only specifies non-zero covariances between a point and itself (i.e. variances); all cross terms (covariances) are zero. Thus the modeled behavior is that there is no correlation between data points. Further, the model terms grow away from data points, while the conditional terms shrink. Consider the combined covariance behavior as defined by (A-16) with these definitions. When interpolation point p lies on top of a data point i, the blending is minimal (i.e. $B_{ip} = 1$) and the model term contributes nothing, as desired. Conversely, when p is far from i, blending is maximal (i.e. $B_{ip} = 0$), and the blended portion of the model term cancels out the conditional covariance (which is also blended), leaving only the unblended estimated background variance from the model. Thus, we have achieved the desired behavior.

Using equations (A-17) through (A-20) and recasting them in terms of correlation coefficients and variances, we get the Bayesian version of (A-14)

$$\sum_{j=1}^{n} w_j \left(B_{ip}B_{jp}\rho_{\Theta_i\Theta_j}\sigma_{\Theta_i}\sigma_{\Theta_j} + \delta_{ij}[1 - B_{ip}B_{jp}]\rho_{\Theta_i\Theta_j}\sigma_{\Theta_i}\sigma_{\Theta_j} + \delta_{ij}B_{ip}B_{jp}\sigma_{\xi_i}\sigma_{\xi_j} + \delta_{ij}[1 - B_{ip}B_{jp}]\sigma_{\xi_i}\sigma_{\xi_j} \right) \quad \text{(EQ A-21)}$$

$$= B_{ip}B_{pp}\rho_{\Theta_p\Theta_i}\sigma_{\Theta_p}\sigma_{\Theta_i} + \delta_{ip}[1 - B_{ip}B_{pp}]\rho_{\Theta_p\Theta_i}\sigma_{\Theta_p}\sigma_{\Theta_i}$$

which with some cancellation of terms can be rewritten as

$$\sum_{j=1}^{n} w_j \left(B_{ip} B_{jp} \rho_{\Theta_i \Theta_j} \sigma_{\Theta_i} \sigma_{\Theta_j} + \delta_{ij} [1 - B_{ip} B_{jp}] \rho_{\Theta_i \Theta_j} \sigma_{\Theta_i} \sigma_{\Theta_j} + \delta_{ij} \sigma_{\xi_i} \sigma_{\xi_j} \right) \qquad \text{(EQ A-22)}$$

$$= B_{ip} B_{pp} \rho_{\Theta_p \Theta_i} \sigma_{\Theta_p} \sigma_{\Theta_i} + \delta_{ip} [1 - B_{ip} B_{pp}] \rho_{\Theta_p \Theta_i} \sigma_{\Theta_p} \sigma_{\Theta_i}$$

This equation is deceptively complex looking: notice that the second two terms on the left hand side are zero except when $i = j$, i.e. on the *diagonal* of the matrix. The off-diagonal terms are described completely by the first term. Similarly, the second term on the right hand side is zero except if $i = p$, i.e. if the interpolation point coincides with one of the data points. Further, if i = j (on the diagonal of the matrix) and/or i = p (interpolation point on top of a data point), additional terms can be cancelled.

Using the same substitutions, we can obtain the Bayesian version of (A-15), the equation for the variance of the error estimate at point p

$$\sigma_{ep}^2 = \sigma_{\Theta_p}^2 - \sum_{i=1}^{n} w_i \left(B_{ip} B_{pp} \rho_{\Theta_p \Theta_i} \sigma_{\Theta_p} \sigma_{\Theta_i} + \delta_{ip} [1 - B_{ip} B_{pp}] \rho_{\Theta_p \Theta_i} \sigma_{\Theta_p} \sigma_{\Theta_i} \right) \qquad \text{(EQ A-23)}$$

Again, the equation appears more complex than it is: the second term in the summation is zero except if i = p, which need not be true for a given point p.

A-2.4 Blending Across Boundaries

We have now satisfied all of the requirements except the fifth. This constraint is solved by imposing new conditions on the blending functions. These conditions impose rapid transitions to the 0 wherever boundaries are crossed. This "patched" transition is illustrated in Figure (A-3) below.

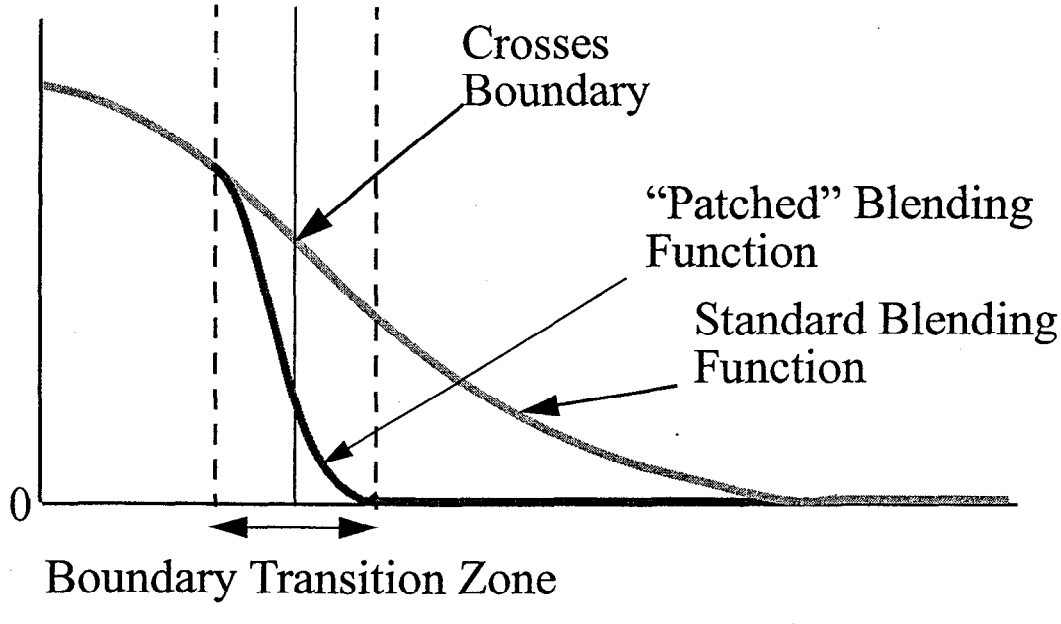


Figure A-3. Typical Region Boundary Blending "Patch"

Using this approach will force residual blending functions to fall to zero as boundaries are crossed. These transitions are continuous and occur over a finite transition zone that occupies both sides of crossed boundary. These transition zones can be made as steep or shallow as the user likes to control the influence that points have on one another if they lie on opposite sides of a defined boundary.

Appendix B: Mesh Refinement

The goal of mesh (tessellation) refinement is to construct a minimal basis for performing fast natural-neighbor interpolations (NNI's) that produce values within a prescribed error bound of the kriged dataset surface. This is accomplished by forming an irregular grid with mesh spacing proportional to the curvature of the kriged surface. The irregular grid need have no more points than are required to ensure a specified level of accuracy in the NNI. Furthermore, the irregular grid is just as easily constructed on the surface of a sphere as it is in a flat plane. For graphical simplicity we illustrate the flat-plane construction technique here.

In this appendix we will begin by describing the mesh refinement methodology in more detail. This discussion will include the primary five step algorithm used to construct a refined tessellation from a previously specified continuous surface (the kriged dataset surface). Next, we will examine the effect on the refined mesh when multiple surfaces are represented on a single tessellation. We shall see that it can have an apparent negative impact on mesh density (increasing) while actually improving the computational capability of libKBI relative to both data access and storage efficiency, and containing triangle search performance. Finally, we shall examine the effect on mesh density caused by modifications in the mesh refinement relative error parameter. But first, lets examine the mesh refinement algorithm in greater detail.

B-1 Mesh Refinement Methodology

The mesh refinement algorithm begins by prescribing a minimal set of nodes that define the boundary of the tessellation and include all of the dataset locations where known values exist (on the surface of a sphere a global grid is constructed which has no boundaries). Once the initial nodes are defined an iterative five step refinement loop is entered that adds nodes to the tessellation until the interpolation error is less than the prescribed accuracy everywhere on the dataset surface.

The five iterative steps are:

1. **Tessellating**
2. **Smoothing**
3. **Kriging**
4. **Curvature Refinement**
5. **Accuracy Refinement**

The first step involves connecting all the nodes together to form a triangular tessellation using a standard Delaunay tessellation technique such as incremental insertion (described in section 2.1.2). Step two involves smoothing the triangles (i.e. moving the nodes, excluding the original data and boundary nodes), using a length-weighted Laplacian smoother to remove poorly formed triangles. Next, (step 3) a

vlaue is assigned to each new node by performing Modified Bayesian Kriging calculation. Then (step 4) all nodes that exceed specified surface roughness requirements are split by inserting a new node at the mid-points of the edges that share the rough node. Finally, (step 5) all remaining triangle edge midpoints and centers are interpolated using a gradient-modified NNI and kriged using the Modified Bayesian Kriging algorithm, after which differences between the two methods are recorded. New nodes are created for insertion into the tessellation at all edge and triangle mid-points for which the differences exceed the specified relative error tolerance between the interpolated and kriged values.

Steps 1 through 5 are repeated in an iterative fashion until all relative error and surface smoothness requirements are satisfied. In the following subsections we shall describe each refinement step in greater detail, including the initialization step, commonly referred to as step 0.

B-1.1 Mesh Initialization (Step 0)

An initial tessellation must be fabricated to act as a starting framework for the refinement process. The initial tessellation contains two sets of nodes. The first is a very coarse regular grid that effectively defines the boundary of the region to be tessellated and prescribes the minimum nodal density to be used throughout the tessellation. All interior nodes (those not on the region boundary) in the coarse mesh are allowed to move throughout the refinement process. The tessellation region's boundary is generally defined as a rectangle (planar definition) and the nodes that define the corners of that rectangle are always fixed. All other boundary nodes are constrained to float along their respective boundary axis only.

The second set of nodes in the initial tessellation is the set of actual data locations used as the basis for creating the Kbase dataset (i.e. the Kriging raw or "ground truth" data). These nodal locations will remain fixed throughout the mesh refinement process. The raw data are included as nodes for two reasons. First, if they are not included and the kriged surface possesses variation much smaller than the average triangle size in the coarse mesh, then it is possible that the curvature and accuracy tests will not sense the minimal variation at their respective discrete test positions (triangle centroids and edge mid-points). In other words, if the spatial range of the surface variation is much smaller than the covering areas of the coarse triangles used to represent the surface, the refinement tests may fail. Since kriging generally forces maximum curvature on or near it's raw data points one can be guaranteed that the initial mesh will capture all of the large magnitude variation if the raw data is included. The second reason concerns the need of researchers and analysts to preserve the original data points. It is easier to recover the values for these points if they are actually assigned to a distinct nodal position within the tessellation. Figure B-1 shows a typical initial nodal distribution before the tessellation is generated that connects the nodes.

Once an initial set of nodes has been defined the algorithm enters the main refinement loop. The first step in this loop is to tessellate the nodes which creates the triangles establishing the mesh connectivity.

B-1.2 Tessellating (Step 1)

Proper implementation of an NNI requires that a Delaunay tessellation be used to form the node-to-node connectivity information (Delaunay, 1934). Recall from section 2.1.2 that we chose an "incremental" insertion algorithm to perform the Delaunay tessellation construction. The incremental algorithm requires that each new node be added one at a time and does not necessitate that the entire tessellation be re-built when new nodes are added at each refinement iteration or when existing nodes are moved in the smoothing step (next section). Figure 5 of section 2.1.2 illustrates the four step process required when "incrementally" inserting a new node into an existing tessellation.

This approach (incremental) lends itself superbly to the mesh refinement algorithm which by its very definition is an incremental node addition algorithm. During this step in each iteration, new nodes created in the previous iteration (during steps 4 and 5 described below) are added to the existing tessellation. Figure B-2 shows an example tessellation just prior and just after insertion of new nodes from the previous iteration. Notice that some of the triangles formed after inserting the new nodes can be quite distorted. The next step in the refinement algorithm is used to help eliminate most of the triangle distortion thereby producing a smooth and uniform change in triangle density throughout the mesh.

B-1.3 Smoothing (Step 2)

Although Delaunay tessellations form the best shaped triangles possible they are formed under the constraint of a prescribed nodal distribution. By modifying the nodal distribution through the use of smoothing techniques, the triangle shape can be improved upon further and the transition in triangle sizes through regions of changing nodal density can be made much smoother. The smoothing process can be accomplished in many ways but the most common (in 2D) is some form of Laplacian smoothing. Laplacian smoothing modifies the coordinates of a particular node by assigning the coordinates to a normalized-weighted linear combination of its nearest neighbor coordinates. The weight function used to multiply the neighbor coordinates can be almost any quantity and for standard Laplacian smoothing the weight function is simply 1 (i.e. the new location is just the average of the neighbors coordinates). Unfortunately, simple weighting is not always robust (convex boundaries are handled improperly) and other weighting forms exist that, in general, do a better job. Typical robust weight functions include node-to-node neighbor length or triangle areas. The mesh refinement algorithm defined here uses the length-weighted Laplacian Smoothing technique (Jones, 1979) exclusively in the smoothing process. The length-weighted Laplacian smoothing for node, $\hat{n}_i$, is defined as

$$\hat{n}_i^* = \left(\sum_{j=1}^{n} L_{ij} \hat{n}_j \right) \Big/ \left(\sum_{j=1}^{n} L_{ij} \right), \qquad \text{(EQ B-24)}$$

where the nodes $\hat{n}_j$ are nearest neighbors of node $\hat{n}_i$ and the lengths (or weights), L_{ij}, are defined by

$$L_{ij} = \left| \hat{n}_i - \hat{n}_j \right| \ , \qquad \text{(EQ B-25)}$$

which are simply the distances between the i'th node and its j neighbors. The superscript (*), in equation B-1 above, indicates that this is an update over the original position used in the weight length calculations (the L_{ij}). Figure B-3 below illustrates the movement of a single node based on a length-weighted Laplacian smooth.

Smoothing is generally a Gauss-Sidel iterative process where each node is updated sequentially using previous smoothing updates of its neighbors. The neighbors themselves may also be recently inserted nodes in need of smoothing. The list of nodes requiring smoothing is looped over several times until the difference between the new and old positions of each node is less than some tolerance. Mathematically we can write

$$\text{iterate over i until} \quad \left| \hat{n}_i^* - \hat{n}_i \right| \le \varepsilon \quad \text{, for all i.} \qquad \text{(EQ B-26)}$$

Figure B-4 shows the example tessellation from Figure B-2 following the new node insertion and the subsequent smoothed mesh after the mesh smoothing step is performed. Notice that most distorted triangles are reformed and some edge connections are modified to force the mesh to adhere to the Delaunay prescription. Also, notice that the fixed data nodes remained fixed and are not modified during the smoothing process. Once the new nodes have been smoothed we are ready to krig those nodes to obtain value and error estimates at each new location.

B-1.4 Kriging (Step 3)

Recall from section 2.1.1 that kriging is a way of calculating values from a dataset at an arbitrary location that also yields an accompanying error estimate. Since the NNI requires these values at all tessellation node points, every new node added during the previous iteration, plus any existing nodes that moved during the previous smoothing step, must be kriged during the current iteration. Note that the entire tessellation is NOT re-kriged during each iterative pass. Only new insertion nodes or existing nodes that moved during smoothing need to be kriged. The resulting kriged value, error, and longitudinal and latitudinal derivatives (formed by kriging extra points and forming differences) are saved for each kriged node.

Although step 3 is the middle step of the mesh refinement iteration, in terms of arriving at a complete tessellation with all nodes determined, tessellation connectivity prescribed, and surface values calculated, it is unequivocally the last step of tessellation construction. The next two steps involve the determination of whether new nodes are to be inserted in the existing tessellation during subsequent refinement iterations. Should the existing tessellation not require any new nodes the current tessellation state becomes the final one. Specifically, when no new nodes are added the algorithm terminates (recall Figure 6) and the tessellation values existing after this step become the final values returned from the mesh refinement implementation (see Appendix E for functional arguments and return values).

In addition to kriging at new and existing (moved) node locations, we shall see that we must also perform the kriging operation during step 5 (accuracy refinement). Those kriging calculations are used to compare against the interpolated values at tessellation edge mid-points and triangle centroids to determine if the kriged surface is accurately represented by the interpolated surface. That process will be defined in greater detail when accuracy refinement is described in section B-1.6 below. Additional information regarding the kriging operation can be found in section 2.1.1 and Appendix A. We are now ready to consider the two node insertion determination steps; curvature and accuracy refinement. We begin with curvature refinement.

B-1.5 Curvature Refinement (Step 4)

Curvature refinement is performed as a pre-step to the accuracy refinement determination. It is performed to minimize the number of points that must be interpolated during accuracy refinement, which can be computationally expensive. The mesh refinement implementation has a parameter toggle that can turn curvature refinement on or off, so a user can construct the tessellation from accuracy refinement decisions alone, if desired (see Appendix E, Section E-3).

Curvature refinement involves examining areas in the surface that are highly peaked (rough) relative to their surrounding neighbors. These areas are generally more likely to exceed the relative error requirements since a substantial second derivative change occurs on the rough nodes (the nodes that lie at the peak). If any rough nodes are discovered we subdivide each edge shared by the rough node and save those locations as new points to be inserted into the mesh in the next refinement iteration. At the same time we remove those edges and triangles from consideration in the next step (the accuracy refinement step). This method is fast, and by selecting the roughness requirement properly, we can reduce the number of edges and triangles that must be interpolated prior to performing the accuracy requirement checks.

Surface roughness is a method of describing the local variability of the data. The roughness is determined by examining the peakedness of a data point relative to its natural-neighbors in the tessellation. Specifically, the roughness r_i for node i is given by

$$r_i = 1 - A_{ip}/A_i, \qquad \text{(EQ B-27)}$$

where A_i is the sum of the areas of the triangles that share node n_i and A_{ip} is the area of those triangles projected parallel to the vector sum of the cross products of each triangle (i.e. the area of the triangles projected onto the average gradient plane through node n_i). The roughness is zero for a node if all of its shared triangles lie in the same plane and approaches one for a triangle group with a high peakedness at their shared node (i.e. the plane of each triangle is nearly perpendicular to the average gradient). Figure B-5 below depicts a typical shallow peak (low r) and high peak (large r) commonly found in generic tessellations.

As mentioned above curvature refinement acts as a pre-step to accuracy refinement. If a user so desires, curvature refinement can be turned off. If this is done accuracy refinement will still produce an accurate grid that meets all stated accuracy requirements described in the next section. However, it has been our experience that the resulting mesh is similar (if not identical) to one generated by combining both refinement methods with the exception that both methods (curvature and accuracy) out-perform the accuracy method alone. This noticeable performance enhancement will generally occur if the curvature refinement criteria (defined as an angle between two adjacent edges sharing a common node) is specified properly. Generally, maximum curvature criteria limits of 30 degrees work well with relative error accuracy requirements of 10% or less. However, depending on the surface undergoing refinement, this is not always a hard and fast rule, and some experimentation on the part of the user may be required to determine an optimum setting.

B-1.6 Accuracy Refinement (Step 5)

The final step in the refinement process is to determine if the current tessellation approximates the kriged surface within a prescribed accuracy requirement. This step enforces the primary condition of the entire mesh refinement algorithm. In order to make this determination we must interpolate and krig at each of the remaining edge mid-points and triangle centroids where surface roughness requirements were not exceeded. To perform the interpolation we use a gradient-modified NNI scheme. This method, like many other interpolation techniques, uses a weighted linear combination of the values at a subset of neighbor nodes in the tessellation. The interpolation method is summarized in section 2.3.4 and is discussed in detail in Appendix D.

Once the necessary locations have been kriged and interpolated using NNI, the final accuracy test can be performed for each. This test is performed as a relative error test of the NNI result relative to the kriged value. If those values differ by more than a prescribed relative amount of the kriged value, then a new node is created for each failed edge mid-point and marked for insertion during the next refinement iteration. This requirement is defined by

$$\left|([Z_k^{krig} - Z_k^{intp}]/Z_k^{krig})\right| \geq \varepsilon, \qquad \text{(EQ B-28)}$$

where Z_k^{intp} is the interpolated value at the kth location, Z_k^{krig} is the kriged value at the k'th location, and ε is the prescribed accuracy tolerance.

B-1.7 Discussion

In the previous sections we have seen that mesh refinement involves defining an initial tessellation that is comprised of a diffuse coarse grid in combination with the raw surface data as a starting mesh. The initial mesh is input into the iterative five step algorithm where it is tessellated and where non-fixed nodes are smoothed to eliminate triangle distortion. The resulting smoothed mesh is kriged at all node locations to form an NNI basis set and is subsequently tested for curvature and accuracy. Those areas (edge mid-points and triangle centroids) which do not pass the tests are split by creating new nodes which are marked for insertion in the next iteration of the refinement scheme.

Note that we begin with a relatively coarse mesh (typically 1 node per 20 square degrees of latitude longitude) and only densify the coarse mesh where improved accuracy is required. However, we have not employed any real optimization technique to ensure that the number of nodes is minimal. In fact, always adding nodes in near proximity to triangles almost guarantees that the mesh will be excessively dense relative to representing the surface to a prescribed level of accuracy. This is because although each edge of a triangle may show interpolated values that exceed the relative error requirement they may, in fact, only slightly exceed it such that the addition and subsequent smoothing of a single node at any one of the edges will fulfill the accuracy requirement.

Techniques that minimize the node density while refining toward a prescribed accuracy do exist (Hoppe, 1995). They typically utilize a penalty function within the framework of a global mesh energy optimization scheme. The penalty function forces the node number to a local minimum while the accuracy requirement necessitates increasing nodal density to meet its requirements. The two functions together in an energy minimization scheme serve to optimize a tessellation relative to both minimizing node density and achieving a relative accuracy criteria. While both elegant and effective toward meeting their respective goals, we felt that such a method would be excessive for the first installment of the mesh refinement algorithm. Currently, we feel that the likely small decrease in mesh density doesn't adequately justify the necessary development time required to implement the algorithm. However, we may pursue this solution in the future.

In the next section we shall discuss the effect of accurately representing multiple surfaces on an individual tessellation. Multiple surface representation is necessary to fully exploit the access methods used by libKBI to retrieve data from the Kbase (See section 2.3.2).

B-2 Multiple Surface Representation

Although the refinement algorithm design implies single surface refinement, by using existing refined tessellations as input (step 0) into the primary five step refinement scheme, other surfaces can also be represented on a single tessellation. For multiple surfaces the algorithm is executed for each consecutive surface using the final mesh calculated for the previous surface as the initial mesh for the next surface. In this way, the necessary mesh density (accuracy) for all previously calculated surfaces is maintained.

As each new surface is added to the mesh the nodal density increases. But, as more and more surfaces are added fewer new nodes are required to meet the accuracy requirements since much of the new surface is already covered with an adequately dense mesh. However, It is important to note that supporting multiple surfaces on a single tessellation will *always* over densify (over refine) the tessellation relative to any one surface used to develop the tessellation. This is because each surface has different regions within which surface variation is maximum. So, where one surface requires increased mesh density to be accurately represented by the NNI, another may have very little variation in that particular area and could have been accurately represented locally with far fewer nodes. The method always guarantees that all surfaces meet the prescribed relative error accuracy criteria everywhere, however, they may possess enhanced accuracy in arbitrary regions throughout the range of the tessellation.

At first glance the major drawbacks to this increased densification would appear to be computational performance. However, it is easy to show that multiple surface representation actually enhances overall performance when client applications access and calculate interpolated values. There are several reasons that lead to this performance benefit and to fully understand them requires that we first examine exactly where imagined performance penalties might occur. Performance penalties could occur in two areas. The first involves a perceived notion that necessary data per tessellation will increase with an accompanying increase in computer resources (memory) required to store the extra data. The second concerns the notion that the time to compute containing triangles (the walking triangle search algorithm) will increase per tessellation due to the increased nodal density.

We don't have to consider NNI as a possible performance bottleneck relative to multiple surface support. This is because we know the NNI performance will be equivalent per tessellation since it is a local method and is insensitive to mesh density once it is initialized with a containing triangle. So the problems with multiple surface support would appear to involve the increased magnitude of data access and subsequent storage, and a performance degradation in the walking triangle search due to the increased nodal density. In fact, it is true that both problems will degrade performance, but, and this is important, the degradation occurs on a per tessellation basis only. One should realize that representing multiple surfaces on individual tessellations reduces the total number of required tessellations to support all surfaces.

Therefore, we need to consider the effects of over densification on a per surface basis which is related to how libKBI clients pose their requests to libKBI. What is the difference in total nodes accessed from individual surfaces, each contained by its own tessellation, to the total nodes accessed from a single tessellation that supports all of the same surfaces? In Figure B-6 we show examples of five surfaces that are fit to the same tessellation at a prescribed relative accuracy of 5%(the surfaces are shown top-down where dark red areas represent regions of positive surface variation and dark blue areas depict regions of negative surface values). Individually each of the surfaces shown in Figure B-6 can be refined (to within 5% relative error of the original kriged surface) to approximately 2000 nodes and 4000 triangles. This adds to a total representation of about 10000 nodes and 20000 triangles for all five surfaces combined. Figure B-6 shows the final tessellation after adding all five surfaces to a *single* tessellation. Observe that the final tessellation contains approximately 4000 nodes and just over 7000 triangles. This means the total savings in nodes and triangles, for this example, is a factor of 3 to 4 when multiple surfaces are refined to a single tessellation. In general, when multiple spatially overlapping surfaces are added to a single tessellation one can expect a savings on the order of the number of surfaces (minus 1 or 2) attached to the tessellation as compared to the case where each surface is defined on its own tessellation.

This result implies that relative to data access and retrieval, multiple surface representation can significantly reduce the necessary data required to define the surfaces within the libKBI environment. Now consider the possible performance degradation that the walking triangle algorithm might experience as a function of increased nodal density due to multiple surface representation. Recall from section 2.3.3 that the walking triangle algorithm possesses an average search time that is proportional to the square root of the number of triangles in the tessellation. This means that if we have, say m surfaces, each assigned to a separate tessellation (m tessellations) then the total search time (t_{single}) required for all m surfaces is

$$t_{single} = mc\sqrt{T} \quad , \qquad \text{(EQ B-29)}$$

where c is a proportionality constant and T is the average number of triangles in each tessellation which we assume to be approximately equivalent.

Now consider the case where one tessellation is used to represent all m surfaces. In this case only one containing triangle search is required but the number of nodes in the tessellation is larger than any one tessellation defined previously that contained a single surface. For purposes of conservatism lets assume that the multiple surface tessellation contains mT triangles. This is extreme conservatism since we know that the tessellation will barely double even after five surfaces are added as illustrated in Figure B-6 above. However, even if our conservative assumption were true then the search time for the single tessellation that represents all m surfaces would be

$$t_{multiple} = c\sqrt{mT} \quad , \qquad \text{(EQ B-30)}$$

which says that t_{single} takes $\sqrt{m}$ times as long as $t_{multiple}$. In reality the actual performance increase for the multiple surface representation method would be much greater since we know that the true tessellation triangle density is much less than mT.

Lastly, recall that the NNI method was not affected by triangle density since it is a purely local procedure and is insensitive to mesh density on a per tessellation basis. However, what effect might the number of tessellations have on NNI performance? As was briefly described in section 2.3.4, and is detailed in Appendix D, the NNI method involves a weight calculation to determine an interpolated result. Although the weight calculation is density insensitive it must be performed for every individual tessellation. This means that the time to perform all NNIs where each surface is contained by a unique tessellation is proportional to m, to borrow from the example above. On the other hand, a multiple surface representation tessellation only requires a single weight calculation which means that single surface representation takes m times as long as multiple surface tessellation representations.

Thus, it is apparent that the multiple surface representation is always the most computationally efficient method and, in addition, requires fewer memory resources than does the case of a single surface per individual tessellation representation.

B-3 Relative Error Effects On Mesh Density

Unlike the multiple surface representation where a definite benefit is received in return for increased mesh density, decreasing the prescribed refinement error (increasing accuracy) results in increased nodal density with an accompanying decrease in computational performance. There is little choice, however, if one needs to more accurately match the actual kriged surface. Just how much accuracy is enough for a particular surface (or group) and how does the increase in accuracy affect node density (performance)?

The answer to the first question involves knowledge of how accurately the kriged surface is known. Typically, the kriged surface is known to the precision of the raw data used to produce the kriged surface to begin with. For example, if the kriged surface is known everywhere to within 10% of the values specified on the raw data then a subsequently constructed refined mesh possessing a relative error of 10% will, at most, reduce the final precision by 1% (10% of 10% is 1%).

As we shall see below, the actual average relative error across the tessellation tends to be about 1/2 of the prescribed relative error criteria. So in reality, we can actually do much better than the example presented above. In general, prescribing relative error accuracies much less than that of the actual raw data does not buy

much in terms of real accuracy relative to the final result. However, increased accuracy definitely effects the number of nodes required to support the requirement. Figure B-7 illustrates the effects of the relative error criteria on mesh density. The example in the figure shows the result of performing mesh refinement on the same surface at three different relative error accuracy criteria of 10%, 5%, and 2.5%. Notice that the node density can increase considerably in areas of high surface curvature that include regions near surface peaks and their respective bases. On average, halving the relative error essentially doubles the numbers of nodes and triangles comprising the refined tessellation.

B-4 Summary

The mesh refinement process is accomplished using a five step algorithm to densify an initial coarse grid representation of a kriged continuous surface to a predefined level of accuracy. The resulting method is capable of refining many surfaces onto a single tessellation. This multiple surface representation on a single tessellation reduces the overall data access and subsequent data storage requirements and enhances performance by improving search times in the walking triangle algorithm. Finally, the method is capable of providing an arbitrary level of user defined accuracy. However, the prescribed magnitude of the chosen relative error criteria should account for the magnitude of the raw data accuracy of the surfaces that the refined mesh will represent and any performance degradation associated with the use of highly accurate (< 5%) surfaces.

(This page intentionally left blank)

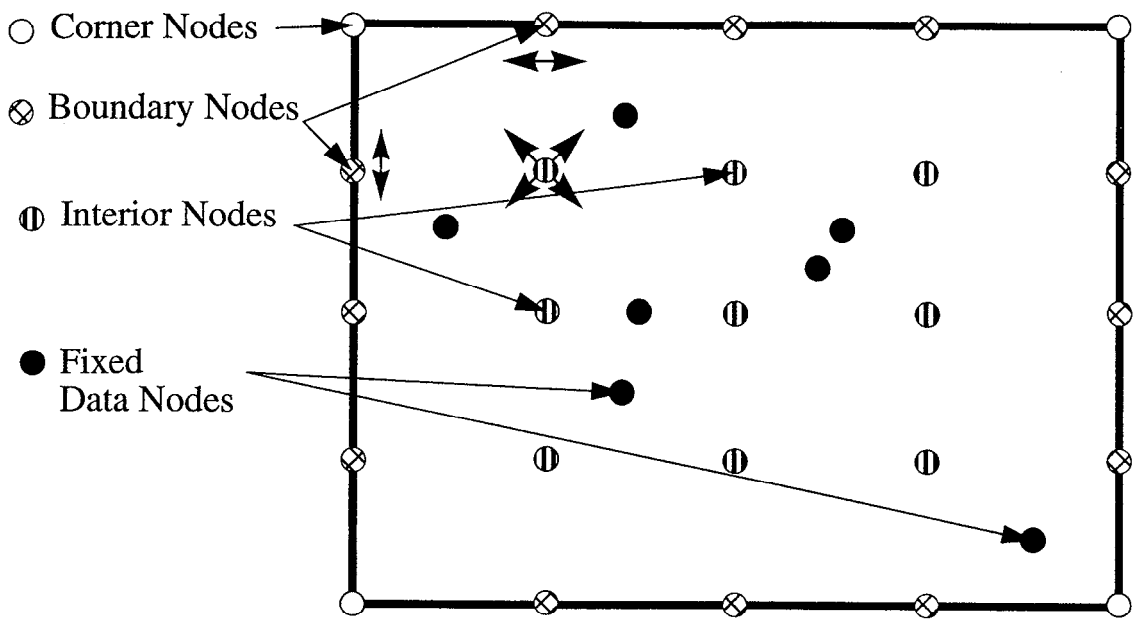


Figure B-1. Initial Mesh Construction (Step 0)
Diagram of a typical initial mesh. Corner nodes are restricted from movement during the refinement process. Boundary nodes are free to move along their defining axis. Interior nodes are free to move everywhere. Data nodes are fixed and never move throughout the refinement process

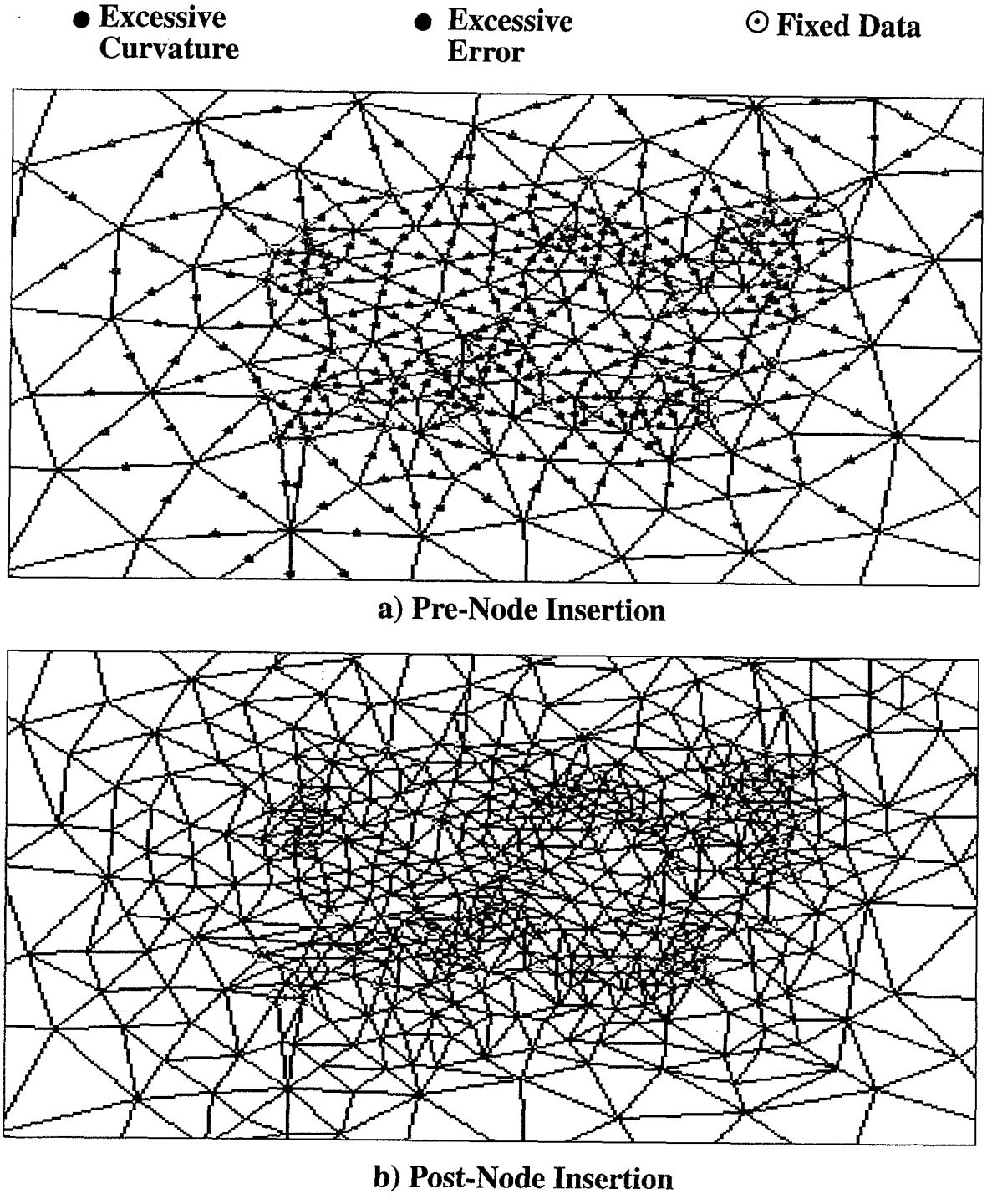


Figure B-2. New Node Insertion And Tessellation (Step 1)
A typical tessellation iteration is depicted illustrating a mesh before node insertion with new node locations marked (a), and immediately following node insertion (b). The new nodes come from the previous curvature and accuracy refinement steps (4 and 5). The fixed data nodes are also shown for clarity.

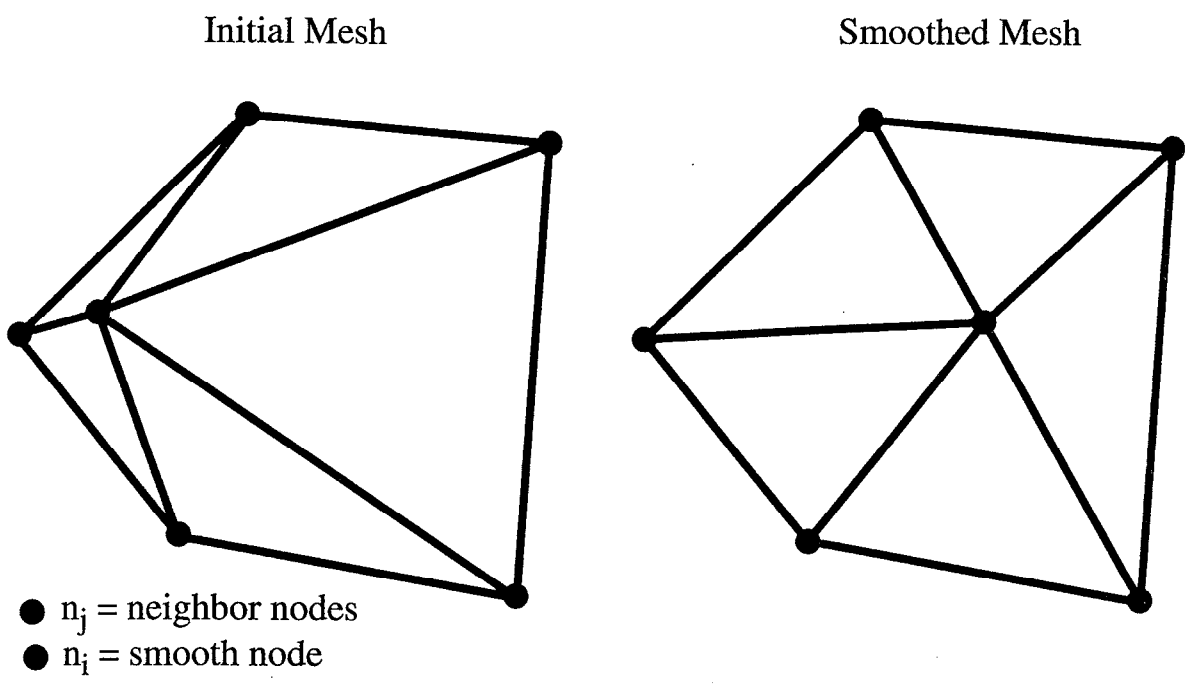
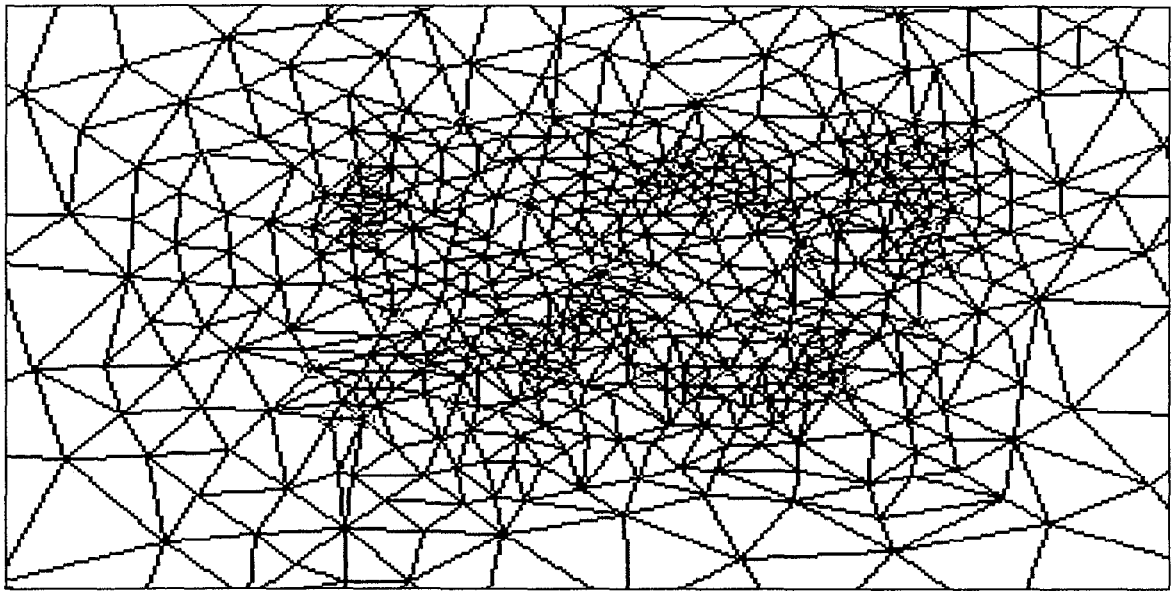
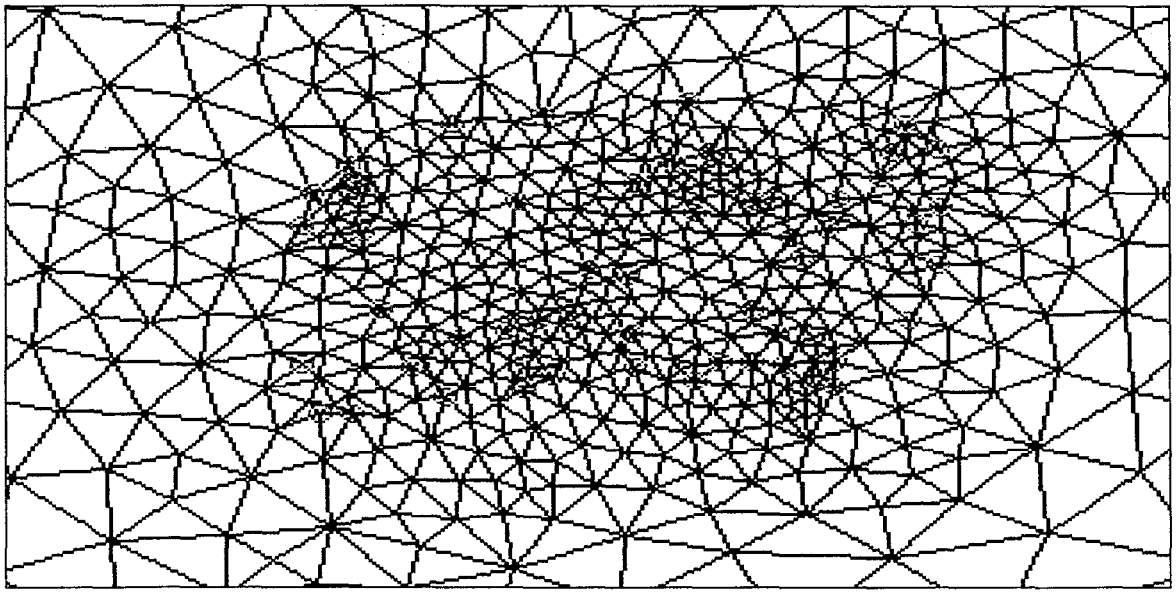


Figure B-3. Length-Weighted Laplacian Smoothing (Step 2)
(left) A set of unsmoothed nodes, (right) the same set of nodes after the center node is smoothed

⊙ Fixed Data



a) Post-Node Insertion



b) Post-Smoothing

Figure B-4. Smoothing (Step 2)
A typical smoothing operation is depicted illustrating the mesh immediately after node insertion (a), and immediately following smoothing (b). Notice that all fixed data nodes have not moved.

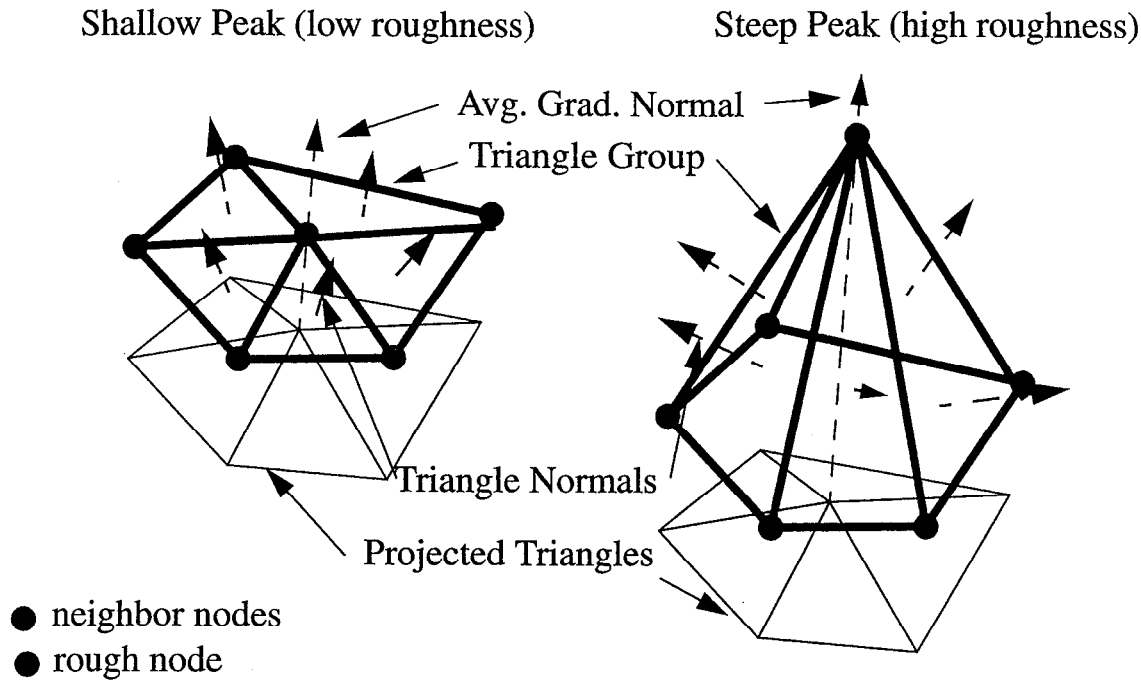


Figure B-5. Triangle Group Surface Roughness
(left) A triangle group that lies essentially in the plane of the groups average gradient, (right) A triangle group that lies largely out of the plane of its average gradient.

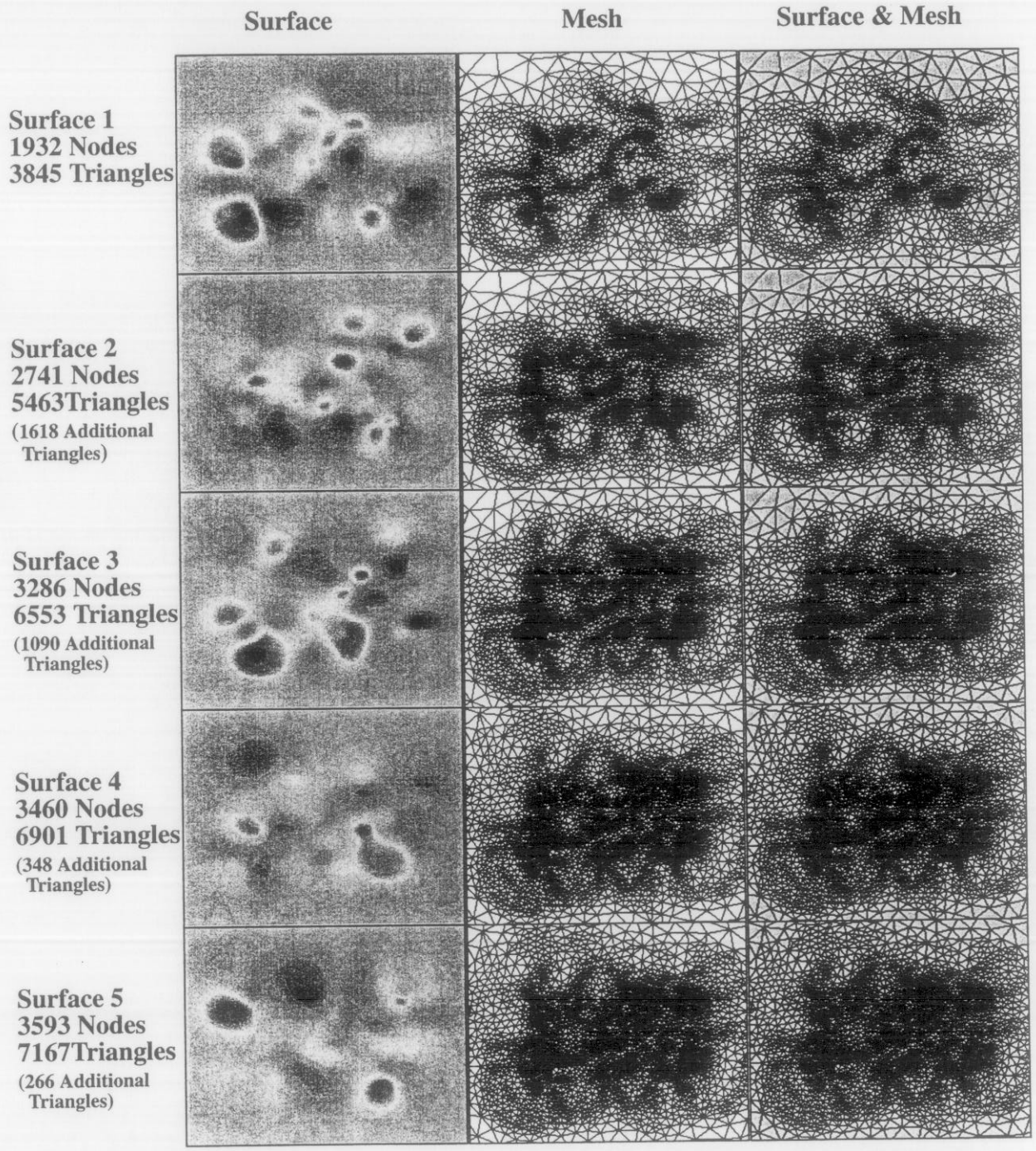


Figure B-6. Multiple Surface Representation On A Single Tessellation
Five surfaces are fit to a single tessellation resulting in approximately a doubling of node and triangle density.

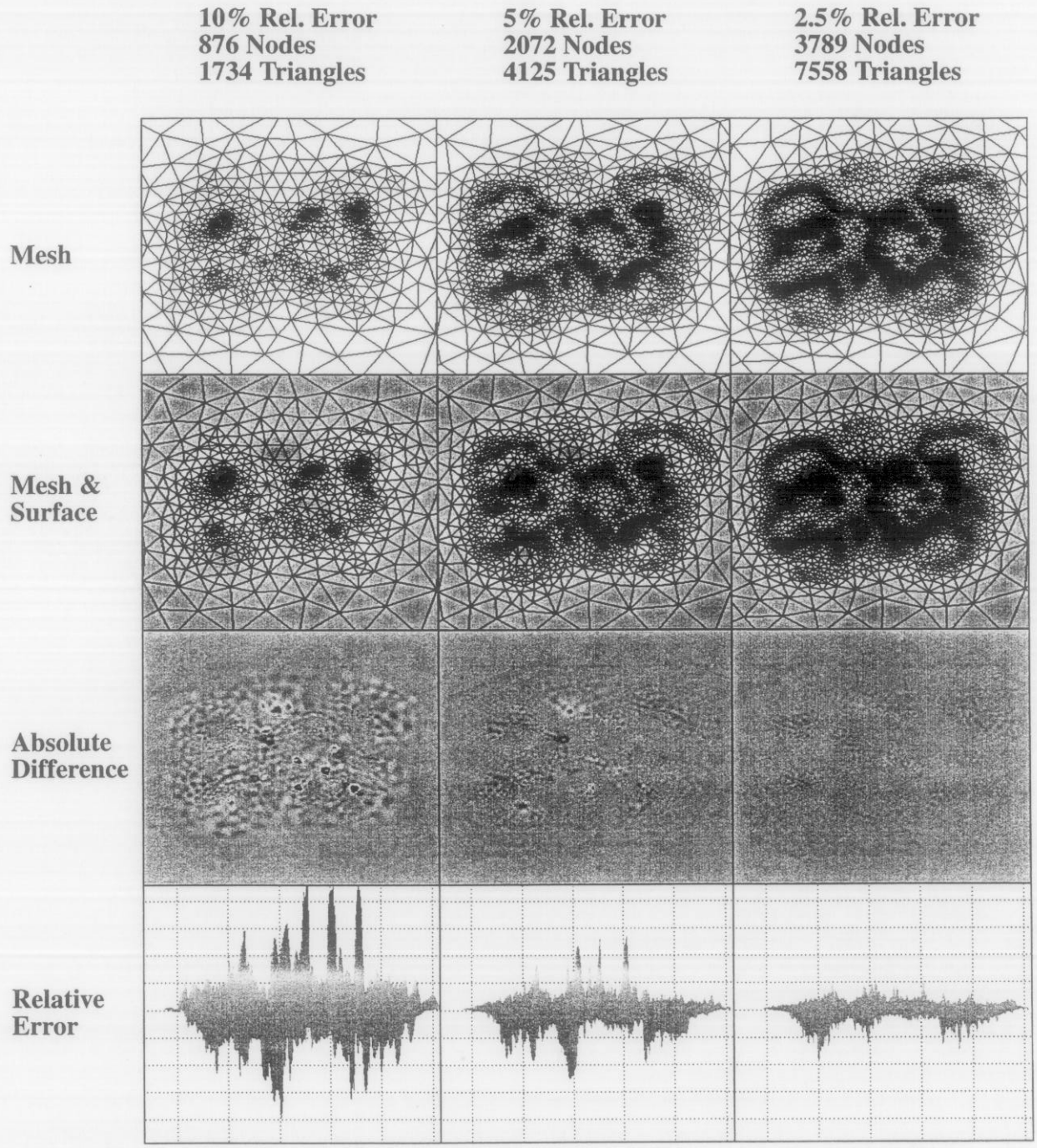


Figure B-7. Relative Error Effect On Mesh Density

In general, mesh node density is inversely proportional to the magnitude of the relative error from which the mesh is refined. The example above illustrates that halving the relative error essentially doubles the numbers of nodes and triangles comprising the refined tessellation. Also, notice that the relative error distribution (the last row of figures) has an average value that is always about half of the relative error criteria. This is caused by over-densification as described in sections B-2 and B-3.

(This page intentionally left blank)

Appendix C: Walking Triangle Search Algorithm

In order to proceed with Natural-Neighbor Interpolation (NNI) we must be able to find the triangle in a given tessellation that completely contains the requested interpolation point. Various methods for rapidly accessing the containing triangle were discussed in section 2.3.3 and reasons for selecting the walking triangle algorithm (Lawson, 1997 or Sambride, 1995) were given. In this appendix we will develop the methodology behind the algorithm and define a minor variation required to perform the search within an arbitrary mesh prescribed on the surface of a sphere.

The walking triangle technique commences by selecting an initial triangle from which tests are conducted to find whether the location of interest lies to the left or right of each edge of the triangle. Each edge of the triangle is treated in succession. If the point falls to the left of an edge, the next edge of the triangle is subjected to the same test. If the point falls to the right, however, the search shifts to the triangle to the right and the process is repeated. Ultimately the containing triangle will be detected when the search point tests left for all three edges of a triangle (i.e. the point is "contained"). In order to accomplish this search we must be able to define what it means for a point to be left or right of a triangle edge. In turn, specifying the concept of spatial "leftness" and "rightness" requires that a strict triangle edge ordering arrangement be maintained.

Figure C-1 illustrates this arrangement for two example triangles that share one of their respective edges. Each edge's direction is defined as counter-clockwise (ccw) around the triangle. A clockwise directional definition would have worked just as well, but the important point is that the definition must be applied *consistently* throughout the tessellation. This means that shared edges between two triangles, as depicted in Figure C-1, have a different direction depending on which triangle is using the edge. So, in the paragraph above when we said that edges of triangles are tested sequentially to determine if a search point lies to the left or the right of one its edges, that means that the edge being tested carries the proper ccw direction relative to the current test triangle.

Now that we know that triangle connectivity is defined by a relative ccw edge ordering, we can define what is meant by "leftness" and "rightness" of a search point relative to an oriented edge. Figure C-2 shows two diagrams, one where the search location is outside the boundary of a triangle, C-2 a), and another where the search point is contained by the triangle, C-2 b). The "leftness" or "rightness" of a point is determined by placing oneself at the beginning node of an edge and facing in the direction of the ccw oriented edge. For example, stand at node n_0 and look in the direction of node n_1. Notice that the search location lies to "your" (the edge's) left. This is also true for the third edge in either diagram (node n_2 directed to n_0). The second edge (node n_1 to n_2), however, has the search point to its right side in the first diagram (a) while it is still to its left side in the second diagram (b). From these observations we can define the condition for containment of a search location by a triangle. A triangle is said to "contain" a point if all of its edges, oriented in a ccw

fashion, see the point to their "left" side. A point is not contained if any of a triangle's ccw ordered edges see the point to their respective "right" side.

We are now able to define a more mathematically rigorous definition of the concepts of "leftness" and "rightness" suitable for implementation within a software algorithm. Figure C-3 shows a directed edge that resides between two separate search locations. The plane that contains the directed edge and the vector normal to the diagram (commonly defined as the z coordinate vector for cartesian planar problems) divides the 2D cartesian space into a "right" side (red in the diagram) and a "left" side (green). The vector from the beginning node of the directed edge (n_0) to its ending node (n_1) is (n_1 - n_0), and the vector from the beginning node to the search point (SP) is (SP - n_0). The cross product of these two vectors will be negative, if the point is left of the directed edge, and positive, if the point is right of the directed edge. Mathematically we can write

$$(SP - n_0) \times (n_1 - n_0) \leq 0 \,, \qquad \text{(EQ C-1)}$$

as the condition for "leftness" of a directed edge relative to some search point. The order of the vectors in the cross product is important; if they are switched the inequality switches. If a point lies exactly on the edge it is assumed to test "left". This means that the first triangle tested that contains a point exactly on it's edge will be determined as the containing triangle. Since either triangle technically contains the point it is irrelevant which one is actually selected for containment.

The result of a typical triangle walk using the concepts described above is illustrated in Figure C-4. The figure depicts the triangle walk for an example tessellation, search point, and initial "guess" triangle, as a series of edge tests until the containing triangle is found. The edge tests begin with any arbitrary edge of the initial "guess" triangle. Each edge is tested for "leftness" or "rightness". In diagram 1) of the figure the first test shows the search point residing to the "left" of the edge. If an edge tests "left", the next edge, proceeding ccw about the triangle, is tested. In the second edge test of the initial triangle the edge test shows the search location as residing "right" of the edge. Any time a search location is found to lie to the "right" the edge is "jumped" to the adjacent triangle that shares the edge just tested. This "jump" effectively moves us closer to the search point. Once the jump is performed the new triangle would find the opposite test result ("leftness") from the shared edge if the edge test were performed again. Recall from Figure C-1, this occurs because the direction of the shared edge must be reversed to maintain ccw behavior in the new triangle. Since the previously tested edge now tests "left" it need not be retested. Therefore, the next edge, adhering to the ccw edge order of the new triangle, is selected for the next edge test.

This process continues until two successive edge tests in the same triangle test "left". Since the edge that was jumped must test "left", two successive edge tests in

the same triangle means all edges test "left", and therefore, the triangle must contain the search location. From this description we can assemble a simple algorithm that defines the walking triangle technique. This algorithm consists of a main function, *triangle_walk*, that takes the search location and an initial "guess" triangle as arguments, and a testing function, *walk_edge_test*, that takes the current edge and triangle objects as arguments and returns a flag indicating "left" or "right" and updated current triangle and edge objects. Triangles, edges, and nodes are treated as objects in the pseudo code, which for purposes of this discussion, implies that at any point in time they represent a single triangle, edge, or node, respectively, and contain all necessary information concerning associated connectivity and neighbor objects. With this in mind the walking triangle algorithm is given as

```
triangle_walk(sp, init_triangle)
{
    assign current triangle, ct, to init_triangle
    find first edge from ct such that the search point, sp, lies to the "left"
    assign the first "left" edge to the current edge, ce

    do forever
    {
        if (walk_edge_test(sp, ct, ce) = "LEFT")
        {
            if (walk_edge_test(sp, ct, ce) = "LEFT")
            {
                ct is containing triangle
                exit and return ct
            }
        }
    }
}

walk_edge_test(sp, ct, ce)
{
    get next ccw edge of current triangle, ct, and assign to current edge, ce
    perform "Left" / "Right" edge test (result in tst)
    if (tst = "RIGHT")
    {
        jump: assign the current triangle, ct, to the adjacent triangle that shares the current edge, ce
        return "RIGHT" (and updated ct and ce objects as arguments)
    }
    else
    {
        return "LEFT" (and updated ce object as an argument)
    }
}
```

The last point to consider before leaving this discussion concerns the representation of a tessellation on a sphere and the subsequent modification required to prop-

erly perform the triangle walk on a 3D surface. Fortunately, the modifications are simple and are restricted to a single aspect of the discussion above, namely the mathematical test required to determine "leftness" and "rightness". For the case of a spherical surface, "leftness" and "rightness" are defined about an arc that passes between two nodes. The arc is a spherical triangle edge that is defined on a great-circle that contains the two nodes and is directed along the path of the great-circle from the beginning node to the ending node. This representation is illustrated in Figure C-5 where the beginning node is defined as n_0 and the ending node is n_1. If we calculate the normal vector of the great-circle we find a common basis from which to perform the "left" / "right" test. By measuring the angle between the search point vector (SP) and the great-circle normal we can determine whether the search point lies to the "left" or the "right" of the great-circle. As shown in Figure C-5, if the angle is less than 90 degrees (the green search point) the search point will lie to the "left" of the great-circle containing the test edge, while angles greater than 90 degrees (the red search point) mean that the search point lies to the "right" of the great-circle. Or in other words, if the dot product of the search point vector with the great-circle normal vector is >= 0 (cos of the angle is less than 90 degrees) then the search point lies to the "left" of the great-circle relative to the orientation of the triangle edge, otherwise it lies to the "right". Mathematically the spherical surface "leftness" criteria is written as

$$SP \bullet (n_0 \times n_1) \geq 0 \quad , \qquad \text{(EQ C-2)}$$

This calculation assumes that the nodes and search point vectors are defined on the unit sphere and that the great-circle normal vector is also normalized before calculating equation C-2. Search points that lie exactly on the great circle test "left" as was analogously described for the planar case.

In summary the walking triangle technique provides a simple method of converting an otherwise costly searching mechanism into a highly efficient location algorithm. The technique utilizes simple triangular mesh connectivity arrangements coupled with trivial geometric vector observations to produce a workable definition of the concept of search location direction and triangle containment. The method is easily implemented within planar cartesian geometry, and with minor modifications, also supports spherical surface geometry.

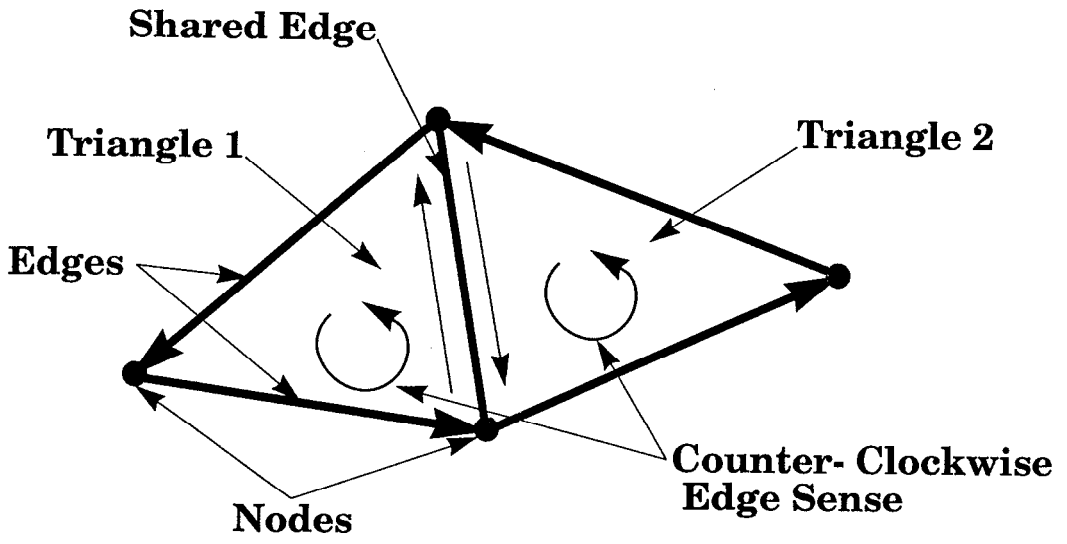
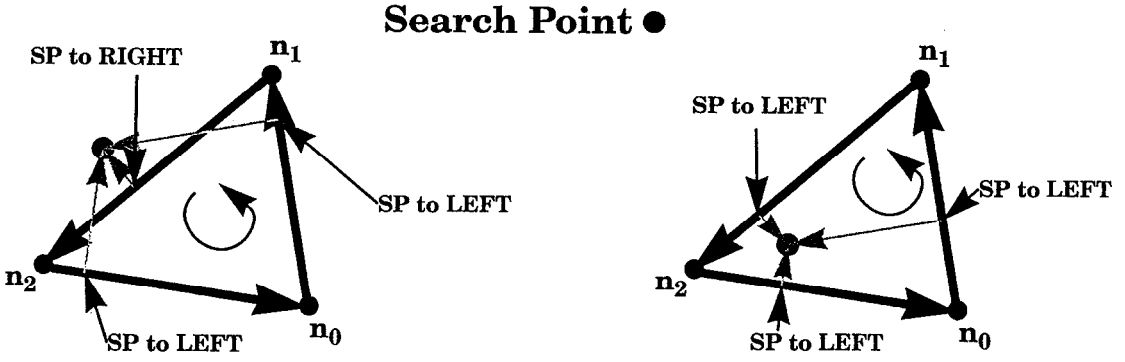


Figure C-1. Triangle Connectivity And Edge Orientation
Triangles are connected by three edges using a node order that is counter-clockwise (ccw) relative to the an outward pointing normal from the triangle face. The shared edge between two triangles has a different directional sense depending on which triangle is viewing the edge. The direction of a shared edge is always defined relative to a triangle such that the ccw edge ordering of the triangle is preserved.



a) Search Point (SP) NOT Contained

b) Search Point (SP) Contained

Figure C-2. Search Point "Left" / "Right" Classification
In diagram (a), the Search Point (SP) lies outside the triangle. This is determined by noting that one of the edges (n1 to n2) shows the SP to its "right" relative to its direction.In diagram (b), the triangle contains the SP because the SP lies to the left of all of the triangles ccw directed edges.

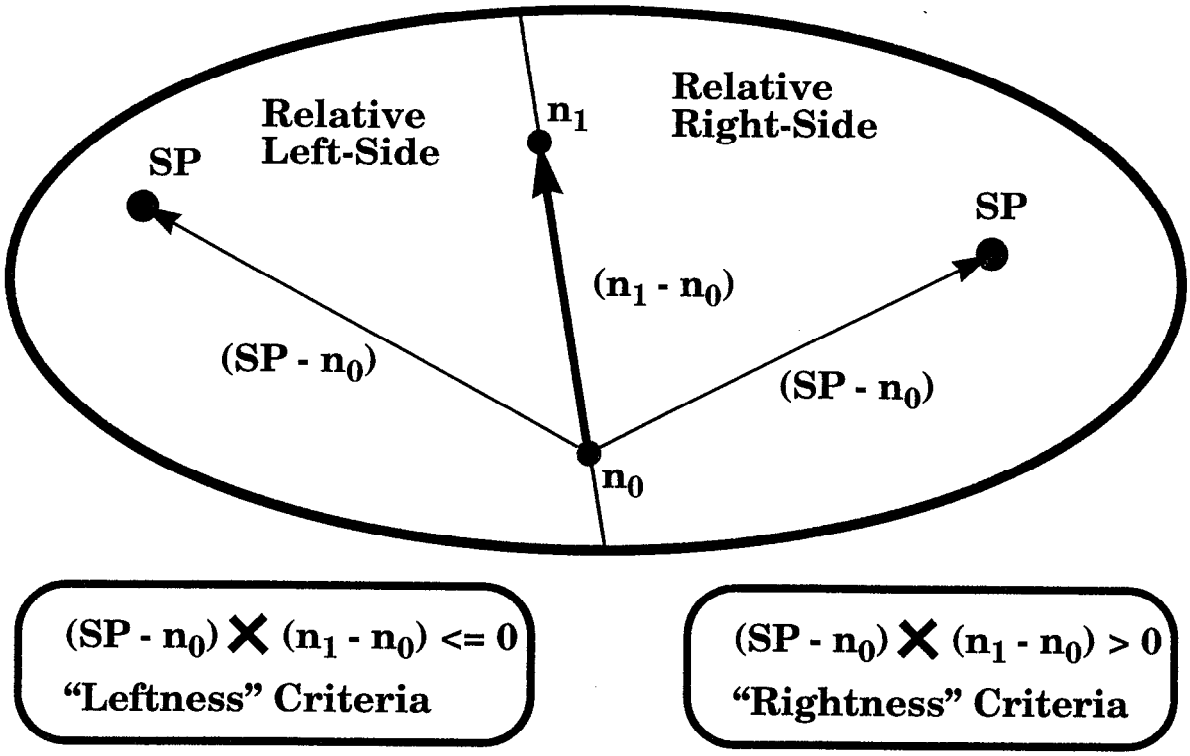


Figure C-3. Edge Criteria For Search Point "Leftness" Or "Rightness"
Mathematically rigorous definitions of "leftness" and "rightness" for planar cartesian directed edges can be found by defining the vector from the beginning node of a directed edge (n_0) to its ending node (n_1), and the vector from the edges beginning node to the search point (SP). The resulting cross product of (SP - n0) with (n1 - n0) will be negative, if the point occupies a position to the "left" side of the directed edge, and positive, if the point resides to the "right" of the directed edge. Points that lie exactly on the edge are assumed to test left (<=). Technically either triangle that shares the edge upon which the search point lies can contain it.

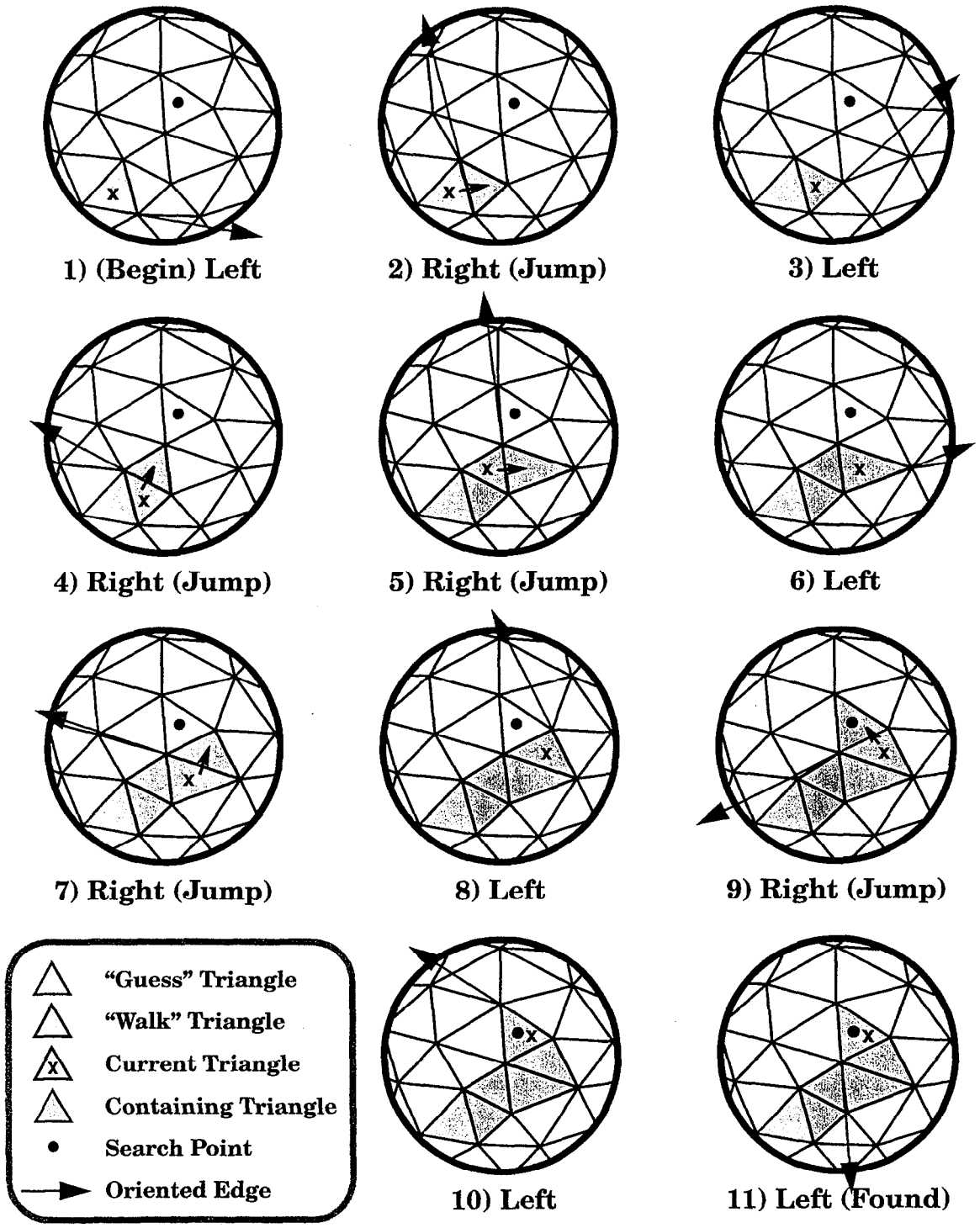


Figure C-4. Walking Triangle Example
This sequence of operations illustrates the walking triangle algorithm. The walk begins at an initial "guess" triangle and tests triangle edges in a counter-clockwise (ccw) sequence. If the search point is to the "left" of the ccw edge, the next edge is tested, if to the "right", a jump into the adjacent triangle is performed. Two successive "left" tests determine the containing triangle.

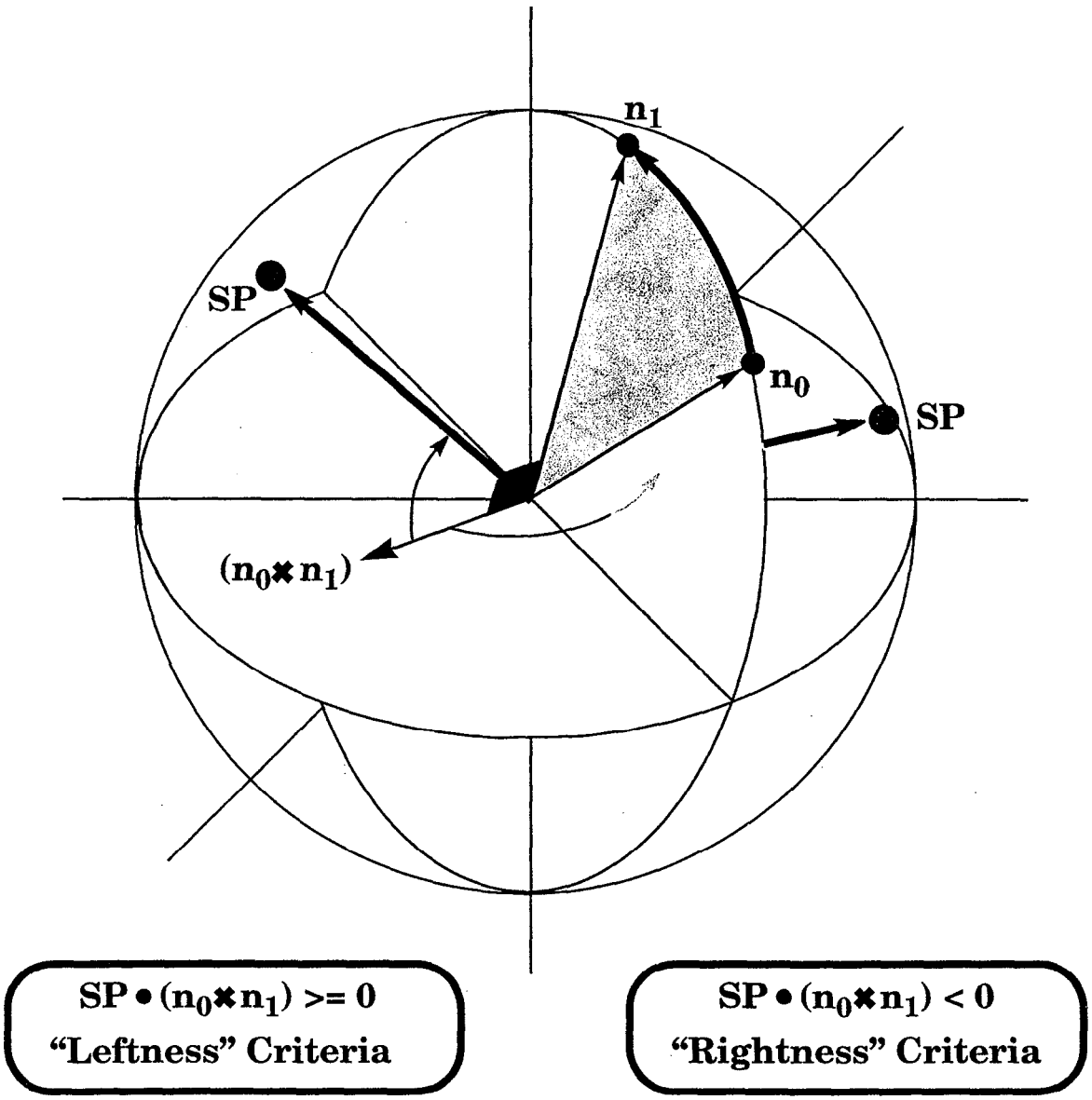


Figure C-5. Spherical Surface "Leftness" and "Rightness" Test
On the surface of a sphere, the"leftness" and "rightness" are defined about an arc that passes between two nodes. The arc is a spherical triangle edge that is defined on a great-circle (the red shaded circle) that contains the two nodes (n_0 and n_1) and is directed along the path of the great-circle from n_0 to n_1. The dot product of the search point vector (SP) with the great-circle normal produces an angular measure (cosine of an angle) that, if positive, categorizes the search point as residing "left" of the great circle relative to the triangle's edge orientation. Similarly, a negative result implies the search point lies to the "right" of the great-circle. This calculation assumes the nodes and search point are defined on a unit sphere and that the great-circle normal is normalized. Search points that lie exactly on the great circle test "left" as was analogously described for the planar case.

Appendix D: Gradient-Modified Natural-Neighbor Interpolation

In section 2.3.4 we discussed the strengths and weaknesses of various interpolation schemes and described how the Natural-Neighbor Interpolation (NNI) scheme was selected because it best satisfied the requirements set forth for the parametric grid interpolator. In this appendix we will develop the methodology behind the scheme including the gradient modification that ensures first and second order differentiability at all tessellation node points.

Recall from section 2.3.4 that NNI is defined as a local weighted average interpolation method which possesses the following form in two dimensions

$$f(x,y) = \sum_{i=1}^{n} w_i(x,y) f_i \quad , \qquad \text{(EQ D-1)}$$

where i is summed over all "natural neighbors" that surround the interpolation point (x, y), f_i is the function value at the nodes, and $w_i(x, y)$ are the normalized "influence" weights attached to neighbors (each i) that influences the interpolation result at (x, y). Defining the methodology behind the determination of the neighbor nodes and their weights for an arbitrary interpolation point within an existing tessellation will be the primary topic in this appendix until the last section.

In the first section below we will discuss the method for determining the Natural-Neighbor (N-N) nodes, edges, and triangles of an arbitrary interpolation point. Recall that these neighbors are found by utilizing the "incremental" insertion algorithm described in section 2.1.2 to treat the interpolation point as if it were a new node to be inserted into the tessellation. This procedure gives us the N-N nodes, edges, and triangles and allows us to define the interpolation point's "insertion polygon" which represents a collection of edges that form an enclosing convex polygon containing the interpolation point.

Armed with this information we shall proceed to describe the weight calculation by first examining the Delaunay tessellation's dual representation, the Voronoi diagram (Voronoi 1908). We shall see that the vertices of the dual Voronoi representation are actually the centers of the tessellation's triangle circum-centers. Recall that the circum-center of a triangle is a circle that uniquely passes through all three of its nodes. We will also find that the "fictitious", or virtual, triangles formed from the assembly of each N-N boundary edge with the interpolation point will possess circum-centers that represent Voronoi vertices of the "fictitious", or virtual, Voronoi polygon defined for the interpolation point. Finally, we shall see that the ratio of areas of the virtual Voronoi polygon to the Voronoi polygons of each N-N node give the necessary weights for equation D-1 above.

Lastly, we will inspect the behavior of equation D-1 when differentiated and evaluated on the nodes of the tessellation. This inspection will reveal that the linear form of NNI, while robust, continuous, and stable throughout the tessellation, is not differentiable at the node points. We will then examine a modification that utilizes the slope of the gradient planes of the N-N triangles to force differentiability at the node points. Accounting for the neighbor gradients at the interpolation point is termed the Gradient-Modified (GM) NNI.

D-1 Natural-Neighbor Determination

In order to calculate the NNI weights for an arbitrary interpolation point we must first determine the appropriate set of N-N nodes, edges, and triangles. For NNI this set is defined as the N-N's of the interpolation point if it were inserted into the tessellation. This means that we need only use the Delaunay "insertion" prescription illustrated in section 2.1.2 to locate the N-N set of the interpolation point. It is important to understand that the interpolation point is not *actually* inserted into the tessellation. Instead, the resulting properties retrieved from a tessellation that contains the interpolation point are used in conjunction with properties from the original tessellation to precisely calculate the N-N objects and their associated weights. We shall demonstrate this process using an example tessellation and interpolation point. This same example will be used exclusively throughout the remainder of this appendix.

Figure D-1 a) depicts the example tessellation and the representative interpolation point. Also shown is the containing triangle which is found from the triangle walk algorithm, and which is necessary to begin the N-N determination. Figure D-1 b) shows the circum-circles of all the surrounding triangles that share any of the three nodes of the containing triangle. Of this set, those that enclose the interpolation point will become the N-N triangles and their respective nodes will become the N-N nodes. Notice, that there are only three triangles and five nodes that belong to the N-N set for the example interpolation point. Generally speaking, 4 to 6 N-N triangles is typical regardless of the location of the interpolation point relative to the Voronoi diagram.

The resulting five N-N nodes in the example are connected by edges that completely enclose the three N-N triangles. These edges are referred to separately as N-N edges, enclosing boundary edges, or simply boundary edges. The same edges are referred to as the enclosing polygon when they are referenced as a group. Each boundary edge is shared by one N-N triangle and another triangle that is not in the set of enclosing N-N triangles. Similarly, edges that are shared by adjacent N-N triangles are called shared edges. Figure D-2 a) illustrates the concepts of boundary and shared edges for the example tessellation/interpolation drawing given in Figure D-1.

The direction of the enclosing polygon's edges are always prescribed in a counterclockwise (ccw) direction (see Appendix C for a discussion of ccw direction). Doing so

preserves the ccw sense of all edges, boundary and shared alike, owned by the N-N triangles. All N-N nodes have exactly two N-N boundary edges that they share. One edge ends on the particular N-N node in question while the other boundary edge emanates from it. The edge that ends on a N-N node is termed the pre-edge while the edge that emanates from a N-N node is called the post-edge. Note that all boundary edges of a N-N enclosing polygon serve as a pre-edge for one N-N node and post-edge for another. This definition is relative to the N-N node being referenced. Figure D-2 b) shows the pre- and post-edge assignment for nodes in the example tessellation. The concepts of pre- and post-edges will be used later when we define the weight calculation algorithm at the end of the next section.

Before we leave the discussion of N-N determination we should briefly examine the result of continuing with the "fictitious insertion" of the interpolation point into the tessellation. Recall from section 2.1.2 that once the enclosing polygon is defined one may remove the N-N triangles, which in effect is the same as removing the shared edges depicted in Figure D-2 a), and insert new triangles that connect the N-N nodes with the new node. In this case the new node is our interpolation point and we really never create the new triangles. They are virtual constructs used solely to extract additional information required to complete the N-N weight calculation. Figure D-3 shows the virtual triangles that connect the interpolation point with the N-N boundary nodes.

D-2 Natural-Neighbor Weight Determination

Having finished with the discussion concerning the determination of natural-neighbor nodes, edges, and triangles we are ready to explore the means by which the N-N weights are calculated. To do this requires that we explain the dual representation of the Delaunay tessellation, the Voronoi diagram (Voronoi, 1908). The Voronoi diagram is a geometric method for defining spatial proximity of an arbitrary collection of points (the nodes in the tessellation to use our example). Each point in the set (the nodes) is uniquely associated with a single representative convex polygon called a Voronoi polygon. Each edge of the Voronoi polygon is called a Voronoi edge and corresponds to an edge of a triangle. Also, each vertex of the Voronoi polygon is called a Voronoi vertex and corresponds to a single triangle in the example tessellation. The ensemble of all such polygons for the entire tessellation is called the Voronoi diagram of the tessellation. The Voronoi diagram has an extensive list of important properties and the reader is directed to (Preparata, 1985 and O'Rourke, 1994) for a detailed discussion. For our purposes only the construction of the Voronoi diagram and its relation to the Delaunay tessellation is important.

A specific Voronoi polygon can be constructed by passing a perpendicular line through the midpoints of the edges shared by a specific node in the Delaunay tessellation. The intersection of those lines form the vertices of the Voronoi polygon. Recall that those vertices are associated with exactly one triangle of the tessellation and, in fact, represent the circum-centers of the circum-circles that inscribe each triangle. Figure D-4 a) illustrates the Voronoi diagram for our example tessellation. The three

containing N-N triangles from Figure D-1 b) are drawn with their circum-circles illustrating their relationship with the Voronoi vertices.

The calculation to evaluate the weights requires that we determine the virtual Voronoi polygon created by the "fictitious" insertion of the interpolation point into the tessellation. The resulting insertion is depicted in Figure D-4 b). Notice that the "fictitious", or virtual, Voronoi polygon has five edges and five vertices. The edges correspond to the five new virtual edges added to connect the interpolation point with the N-N nodes. The five virtual Voronoi vertices correspond to the five additional virtual triangles circum-centers.

The weights for each N-N node are determined by calculating the amount of overlap between the existing Voronoi polygons and the new virtual Voronoi polygon. Figure D-5 illustrates the resulting overlap with the original Voronoi diagram drawn in blue and the new virtual Voronoi polygon for the interpolation point drawn in red. Given that we can now construct the Voronoi diagram of the tessellation and the virtual Voronoi polygon of the interpolation point we are ready to consider the solution technique used to calculate the weights.

The method used by Sambridge (et. al. 1995) to calculate the areas of the overlapping regions of the Voronoi polygon is the same one defined by Watson (1992) which essentially sums the overlapping regions as a group of signed triangles. This method uses the "fictitious" Voronoi vertices (the red circles in Figure D-5), the original Voronoi vertices (the blue circles), and circum-centers of triangles formed from the nodes of the shared edges (Figure D-2) combined with the interpolation point, as vertices of component triangles whose proper summation determines the areas of the individual overlapping areas. Using this method a set of three signed sub-triangles is formed for each N-N triangle. For our example this constitutes nine separate sub-triangles.

Exploiting some efficient indexing techniques while summing the areas of these positive and negative sub-triangle areas results in the sought after weight required by each N-N node. Figure D-6 shows three such sub-triangles, each coming from the analysis of a different N-N triangle. One of the components is positive (the large green triangle in the first panel) while the other two have negative areas (the grey triangles of the second panel). Based on the indexing techniques described in the Sambridge paper, the three sub-triangles are summed to form the weight for node n_3. Notice that the triangles are constructed by using the circum-centers of triangles formed between the nodes of the shared edges and the interpolation point (the green circles in Figure D-6). In general, this method is fast and efficient and will quickly evaluate the necessary areas (weights) for each N-N node.

Unfortunately, this method is indeterminate when the interpolation point lies exactly on one of the shared edges of the N-N triangles. When this happens the nodes of the shared edge and the interpolation point lie on a line and the resulting circum-center is no longer defined (resides at infinity). When this happens the sub-

triangle area is undefined and we are unable to define the necessary weights for nodes that use the undefined sub-triangle. For the work performed by Sambridge (finite-element integration) and others this was not a problem as they only interpolated points that were well within the boundaries of each triangle. Unfortunately, for our application we cannot guarantee that the interpolation point will never lie on a triangle edge and must, therefore, find another way of determining the necessary areas.

Since the problem occurs because the shared edges are used in combination with the interpolation point we propose to calculate the weights without them. We know we can do this because the magnitudes of the five weights can be completely determined from the set of Voronoi vertices that define the virtual Voronoi polygon and the Voronoi vertices obtained from the circum-circles of the N-N triangles. No other point information is necessary to determine the weights. Now all we need is a prescription for performing the calculation. To do this we need to examine the relationships of the circum-centers with respect to the original tessellation's N-N triangles and edges.

Recall that each "fictitious" Voronoi vertex was determined as the circum-center of a triangle constructed from the two nodes of one boundary edge and the interpolation point. The only unique component of the construction that belongs to the original tessellation is the boundary edge. The resulting matching of all of the virtual Voronoi vertices with a specific boundary edge is shown in Figure D-7. Given an arbitrary index for each boundary edge, e_i, the matching Voronoi vertex is labeled ${}^{e}v_i$. Also shown are the Voronoi vertices created from each N-N triangle. These are designated as ${}^{t}v_j$ for the jth triangle. Given these definitions and the definition of the pre- and post-edge we are ready to examine the algorithm that will yield the N-N node weights.

In our algorithm we determine the weight assigned to each N-N node in succession by following the enclosing polygon in a ccw direction. This will involve the determination of several sub-triangles, similar to the method described above, except that the triangles are specific components of a particular N-N node weight. To avoid confusing this method with the previous one, and because in a strict sense these are not sub-triangles of the N-N triangles, we shall refer to the N-N node sub-triangles as "component" triangles.

This process is illustrated in Figure D-8. The node that we begin with is arbitrary. For each N-N node we find it's pre-edge and post-edge and we take the virtual Voronoi vertex assigned to the pre-edge and anchor it as the initial point for all subsequent component triangle area calculations. Next, we follow the current N-N node's Voronoi polygon in a clockwise direction until we arrive at the Voronoi vertex of an N-N triangle that has the pre-edge as one of its edges. This will be the second point of the first component triangle. Finally, we continue to follow the N-N node's Voronoi polygon in a clockwise direction until we encounter the Voronoi vertex of the N-N triangle that shares an edge with the previous N-N triangle. This forms the

third point of the component triangle of the N-N node whose weight is being calculated currently. We can now calculate the area of the component triangle from the relation

$$\frac{\left|(p_3 - p_1) \times (p_2 - p_1)\right|}{2}, \qquad \text{(EQ D-2)}$$

where p_i represents the Voronoi vertices described above and are labeled in Figure D-8 as **"1"**, **"2"**, and **"3"**, respectively. Each panel in Figure D-8 shows a particular component triangle definition. The point labels are ordered such that equation D-2 is always positive.

The next component triangle for a particular node is found by making the old p_3 be the new p_2 and finding a new p_3 by continuing the traversal of the node's Voronoi polygon in a clockwise direction until the Voronoi vertex of the next N-N triangle that shares an edge with the previous N-N triangle is found. This point is subsequently assigned to point p_3. If, however, the next triangle is not a N-N triangle, as is the case when the edge shared by the previous triangle is a boundary edge (the post-edge) then we assign p_3 to the virtual Voronoi vertex owned by the post-edge. When this happens we will be calculating the last component triangle for the current N-N node being processed. This can happen for the first component triangle calculated for a N-N node, as is shown for nodes n_2 and n_4 (panels e) and i) in Figure D-8), or after several component triangles have been evaluated, as is illustrated for nodes n_0, n_1, and n_3. The number of component triangles evaluated for a specific N-N node is equal to the number of the Voronoi vertices owned by the N-N triangles that are defined as vertices in the respective node's Voronoi polygon. For our example, n_0 and n_1 have 2, n_2 and n_4 have 1, and n_3 has 3. This fact is unchanged, as one would expect, from the method (Sambridge et. al.) described above.

Once a particular weight has been evaluated for a specific N-N node we can proceed to the next node weight calculation. This is done by assigning the previous N-N node's post-edge as the current pre-edge, by retrieving the new node's post-edge and its associated virtual Voronoi vertex, and by setting the new anchor point, p_1, to be the old point p_3 from the last evaluation (point p_2 remains unchanged). We are now set to continue finding the third point for successive component triangles for the next N-N node. This process continues until all component triangles for each N-N node have been determined.

Successive component triangles for a specific N-N node are summed to obtain the total area of the overlapping region for that node's weight. The individual N-N weights are also summed to obtain the total area of the virtual Voronoi polygon. This is used to normalize each individual N-N node weight. Since the nearest-neighbor coordinates are normalized they always lie between 0 and 1. This results in the prop-

erty that if an interpolation point falls exactly on top of a node the function value of the node is returned.

Given the description above we can assemble a pseudo-code algorithm to evaluate the N-N node weights. In a similar fashion to the triangle walk algorithm described in Appendix C, we use the abstraction of node, edge, and triangle objects. These objects represent a specific instance of a tessellation member. When the term "update" is mentioned below relative to a specific object (e.g. a triangle) the reader should interpret the term as referring to the fact that the object instance has been reset to another object (e.g. the object now refers to the adjacent triangle that shares an edge with the previous triangle). With this in mind the NNI weight calculation is given by:

```
pick the first node in the N-N node list as the current node, cnode
get the current nodes (cnode) pre-edge as cedge
get the circum-center ("virtual Voronoi vertex") of cedge as ip1
get the owning triangle of the pre-edge (cedge) and assign it as the current triangle, ctri
get the circum-center of ctri as ip2
assign total nodes to be processed to nodecount

totalarea = 0
loopcount = 0
do while (loopcount < nodecount) "loop thru all N-N nodes"
{
    nodearea = 0
    do while (cedge is NOT a post-edge of cnode) "loop thru component triangles of current node"
    {
        get the next ccw edge of ctri, and assign to cedge "cedge change"
        get adjacent triangle of ctri across cedge and assign to adjtri
        if (adjtri is a N-N triangle) then
        {
            assign adjtri circum-center to ip3
            assign ctri to adjtri "ctri change"
        }
        else
        {
            cedge is a boundary edge (post-edge of cnode), get the circum-center of cedge as ip3
        }
        perform triangle area calculation between points ip1, ip2, and ip3, and add to nodearea
                    nodearea = nodearea + (ip3 - ip1) x (ip2 - ip1) / 2
        assign ip2 = ip3
    }
```

assign *cnode* **weight,** *cnode.weight* = *nodearea*

increment total area, *totalarea* = *totalarea* + *nodearea*

assign *ip1* = *ip3*

assign *cnode* **as the opposite node of** *cedge* *"cnode change"*

loopcount = *loopcount* + 1

}

normalize each n-n node weight by dividing by *totalarea*

This method of weight calculation has about the same compuational cost as the Sambridge method described above. The major difference is that this method is connectivity based (e.g. triangles have to know their defining edges and edges have to know what triangles they share) which requires a couple more decisions that aren't always necessary in a purely indexed based technique. On the other hand, the method no longer requires the use of the shared edge circum-circles and is therefore a very robust implementation.

D-3 Gradient Modification

Natural-neighbor interpolation produces surfaces that are continuous but not differentiable at the node points (Sibson, 1980) (Note: the surface produced by kriging has similar properties if the measurement error prescribed for a data point is zero). Since differentiability is a requirement for many of the data sets produced for the Kbase we use a modified version of natural neighbor interpolation that uses the slope of the average gradient (Watson, 1992) at each node of the tessellation to enforce first and second order differentiability. This method forces a slope that is equal to the gradient of the kriged surface at each of the tessellations nodes. The method is termed the Gradient-Modified (GM) NNI.

The method requires that the average gradient slope defined at each N-N node be blended with the linear NNI result. This involves summing the difference between the linear NNI value and each of the values found on the N-N node's average gradient planes evaluated at the interpolation point. Lets call this result the altitude of the i'th node's gradient plane, $S_i(x, y)$, where x and y are the interpolation point's position in 2D. This value can be found from the expression

$$S_i(x, y) = f_i + (x - {}_x n_i)\,{}_x n'_i + (y - {}_y n_i)\,{}_y n'_i \ , \qquad \text{(EQ D-3)}$$

where f_i is the surface value defined at the position of node i, ${}_x n_i$ and ${}_y n_i$ are the x and y positions of the i'th node, and, ${}_x n'_i$ and ${}_y n'_i$ are the derivative components of the average gradient plane that passes through node i.

We can now add to the linear NNI result the sum of the differences of the altitude of the i'th gradient plane with the linear NNI result multiplied by a modifying blending function that reduces the influence far away from a node and increases the influence of these differences when the interpolation point draws nearer to a specific node (the difference operation itself has the opposite properties since $S_i(x, y)$ becomes $f(x, y)$ at each node point). The blending function is generally a normalized function that varies between 0 and 1 and is dependent upon the nodes weight and possibly its surface roughness. The expression for the modified gradient interpolant can be written as

$$f_{gm}(x, y) = f(x, y) + \sum_{i=1}^{n} h_i(w_i, r_i)(S_i(x, y) - f(x, y)) \quad , \qquad \text{(EQ D-4)}$$

where $f(x, y)$ is the linear NNI result and $h_i(w_i, r_i)$ is the blending function that is defined as dependent on the i'th node's weight (w_i) and surface roughness (r_i).

Generally, as a node's weight tends toward zero the influence of the difference in the node's gradient altitude and the linear NNI is reduced to zero. Also, as the nodes weight moves closer to one, the influence of the difference becomes more important and so the blending function's value also moves closer to one. Figure D-9 shows a typical blending function as a function of node weight. The influence of the surface roughness can also be factored in by changing the slope of the blending function for function values evaluated at high node weights (near 1).

The overall effect of equation D-4 is to distribute the difference in the slopes between the estimated gradient and the linearly interpolated basis surface over a small region about each node point. This is achieved by varying the influence of the gradient from 1 at the node point to zero where the node ceases to be a neighbor of the interpolation point. This approach causes the solution to be bounded by the linear basis surface and local trend of the gradient surfaces. The method ensures that the resulting blended surface is smooth everywhere, including the node points.

Figure D-10 illustrates the effect of the gradient modification on the linear NNI method. Panel a) shows a typical highly peaked node using standard linear NNI. Notice that, although continuous everywhere, the surface is not differentiable at the node position (differentiability does exist everywhere else, however). Panel b) shows the same surface after applying the GM method using the blending function shown in Figure D-9. Notice that differentiability is restored to the node location and the peakedness is reduced to an almost spherical appearance.

The GM variant of NNI provides an inexpensive means of restoring first and second order continuity while providing a mechanism for controlling the surface tautness (or peakedness) at each node point. The resulting surface generated by the GM NNI scheme is extremely general, local in scope, and always stable, regardless of the complexity or sparseness of the data distribution.

(This page intentionally left blank)

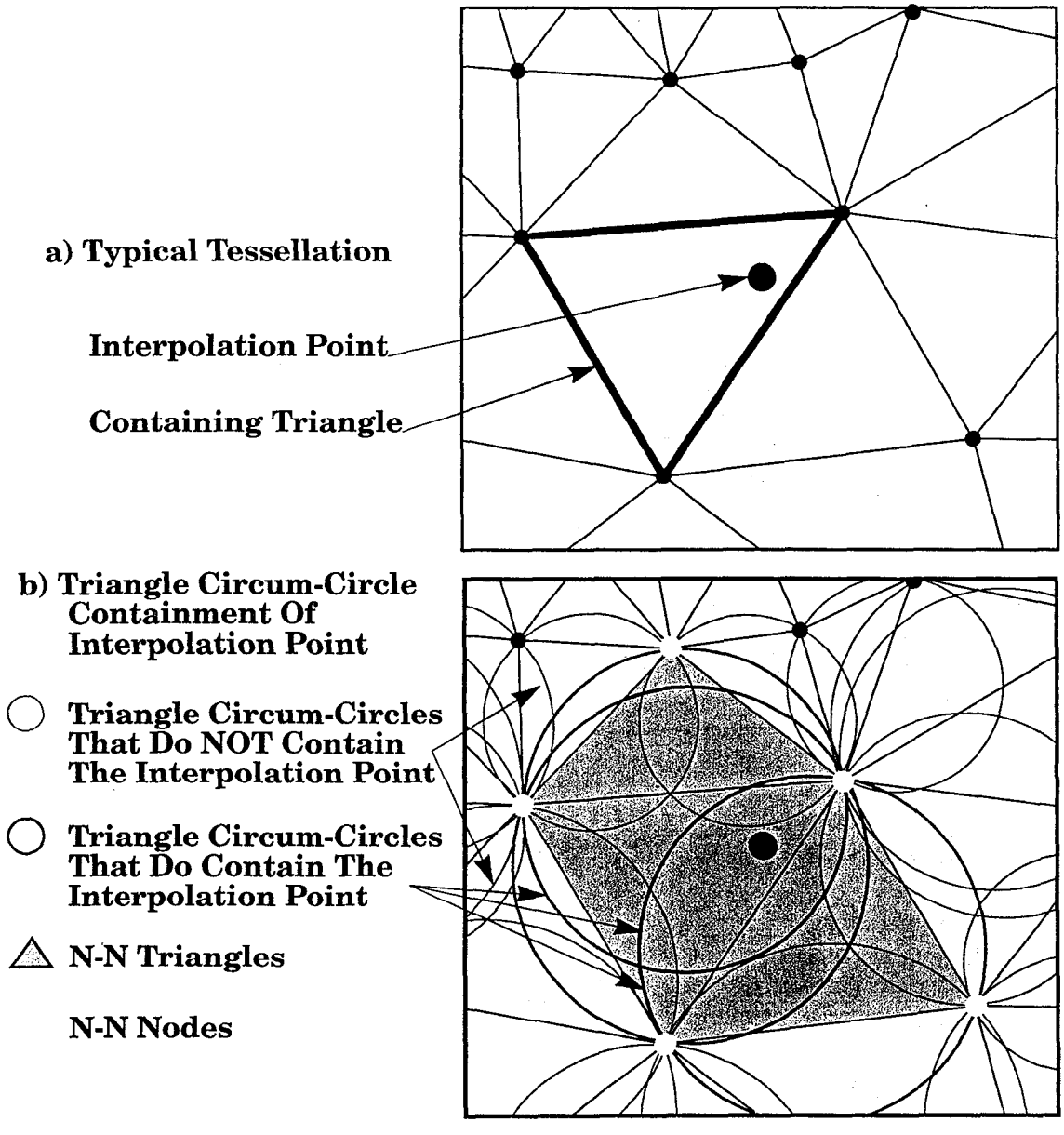


Figure D-1. Interpolation Points Natural-Neighbor Nodes And Triangles
The top drawing, a), illustrates a section of a typical tessellation and an interpolation point. The containing triangle is highlighted with red edges. The bottom drawing, b), shows all of the triangle circum-circles that share any of the three nodes of the containing triangle. Notice, that there are only three triangles and five nodes that belong to the N-N set for the interpolation point in question.

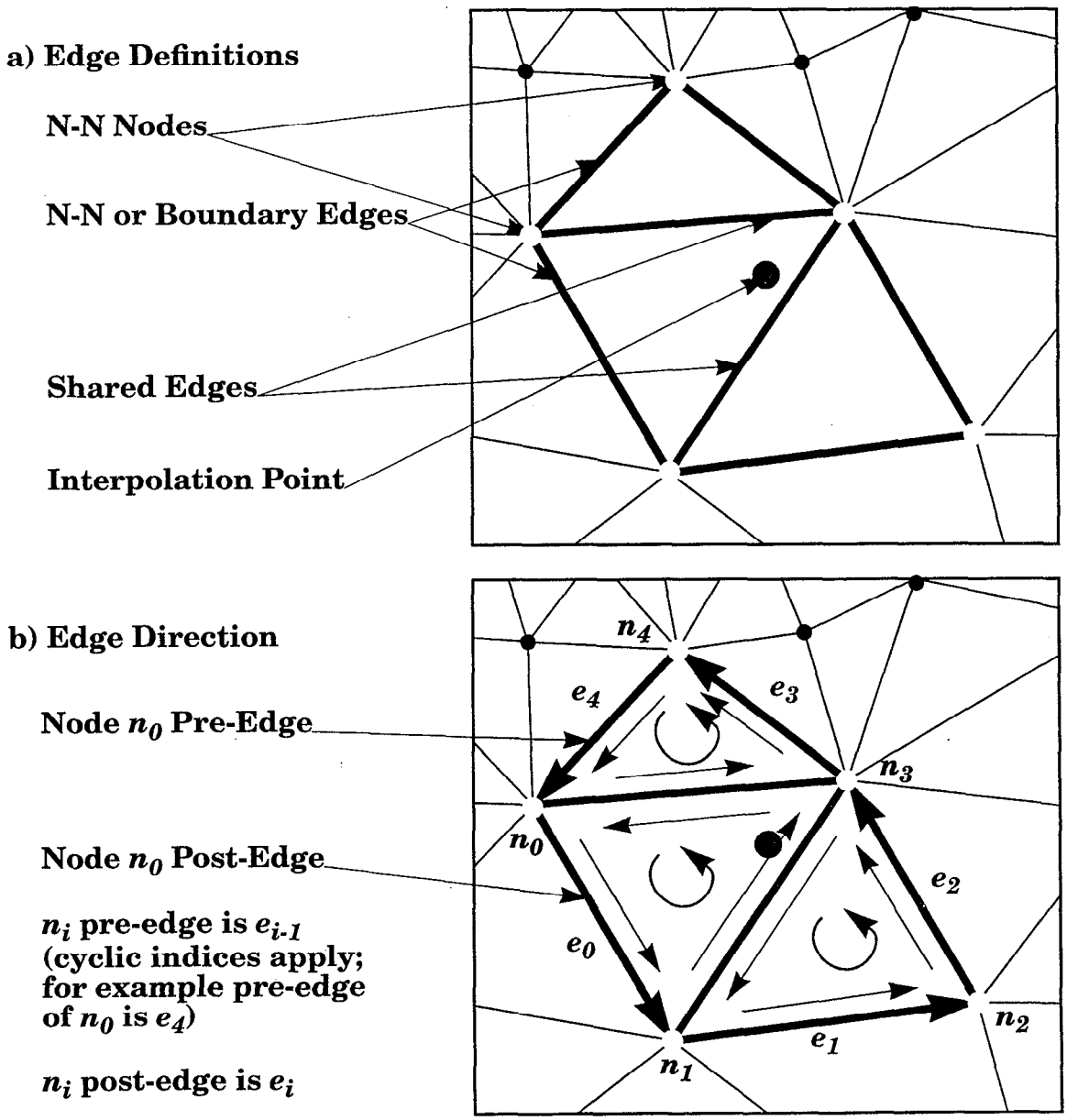


Figure D-2. Natural-Neighbor Edge Definitions And Ordering
a) This Illustrates the concept of N-N edges or boundary edges that connect the N-N nodes to form an enclosing polygon about the interpolation point. The boundary edge can also be defined as an edge that only has one N-N triangle attached. The other triangle is NOT an N-N triangle. A shared edge on the other hand, is shared by two N-N triangles. b) Edge direction is ordered to preserve counter-clockwise (ccw) direction for all N-N triangles. Pre-edges and post-edges are defined relative to each node. An edge is both a pre- and a post-edge for two different nodes. Pre-edges are those that end on a node, relative to the node, while post-edges emanate from their relative node.

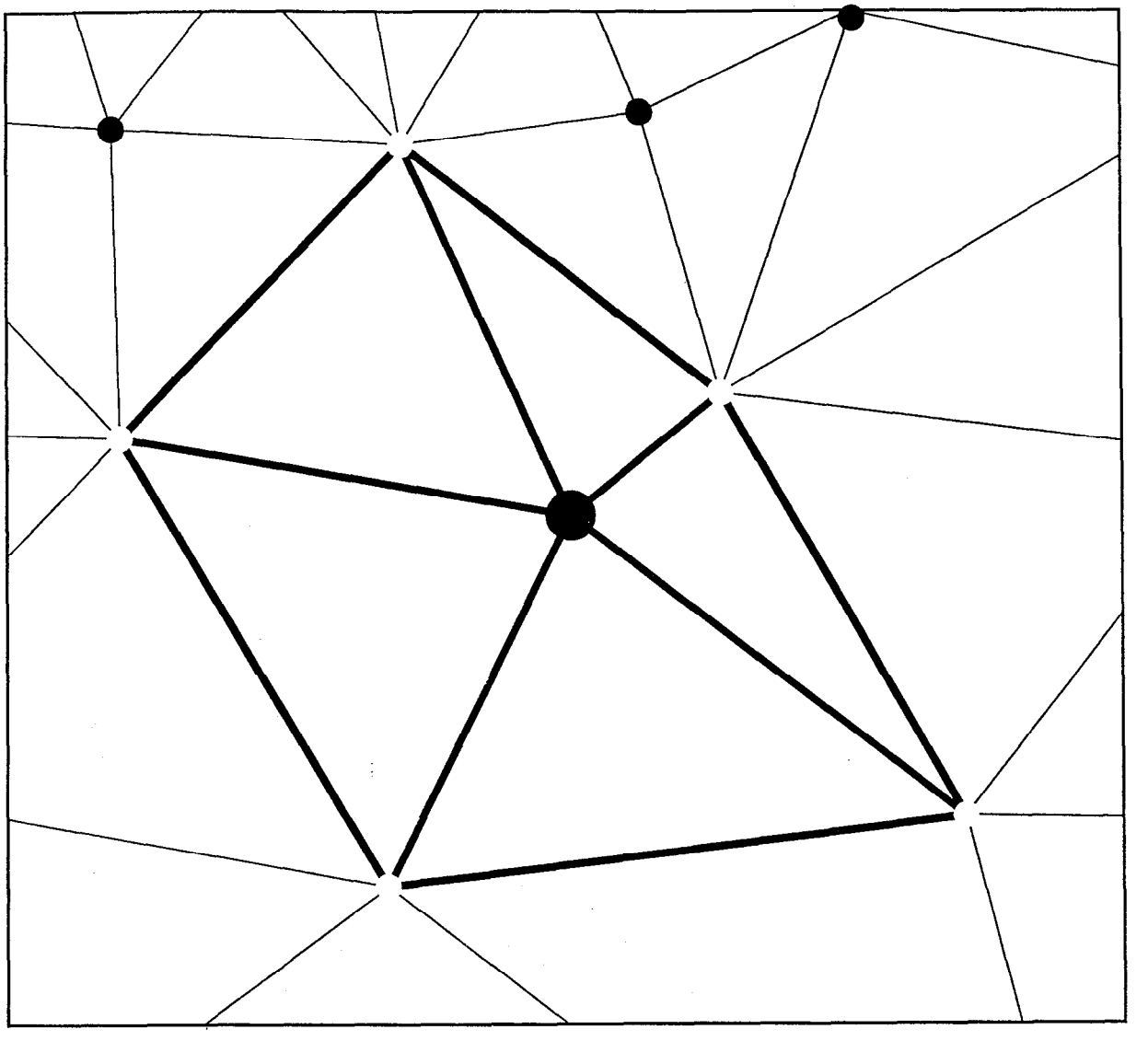


Figure D-3. Virtual Interpolation Point Triangles

If the N-N triangles are removed the interpolation point can be treated as a new node to be inserted in the tessellation by connecting the interpolation point as a triangle vertex to each of the pair of nodes that define a separate boundary edge. These triangles are "fictitious" and are not created. However, the circum-centers from the circles that enclose each virtual triangle are used in the weight calculation.

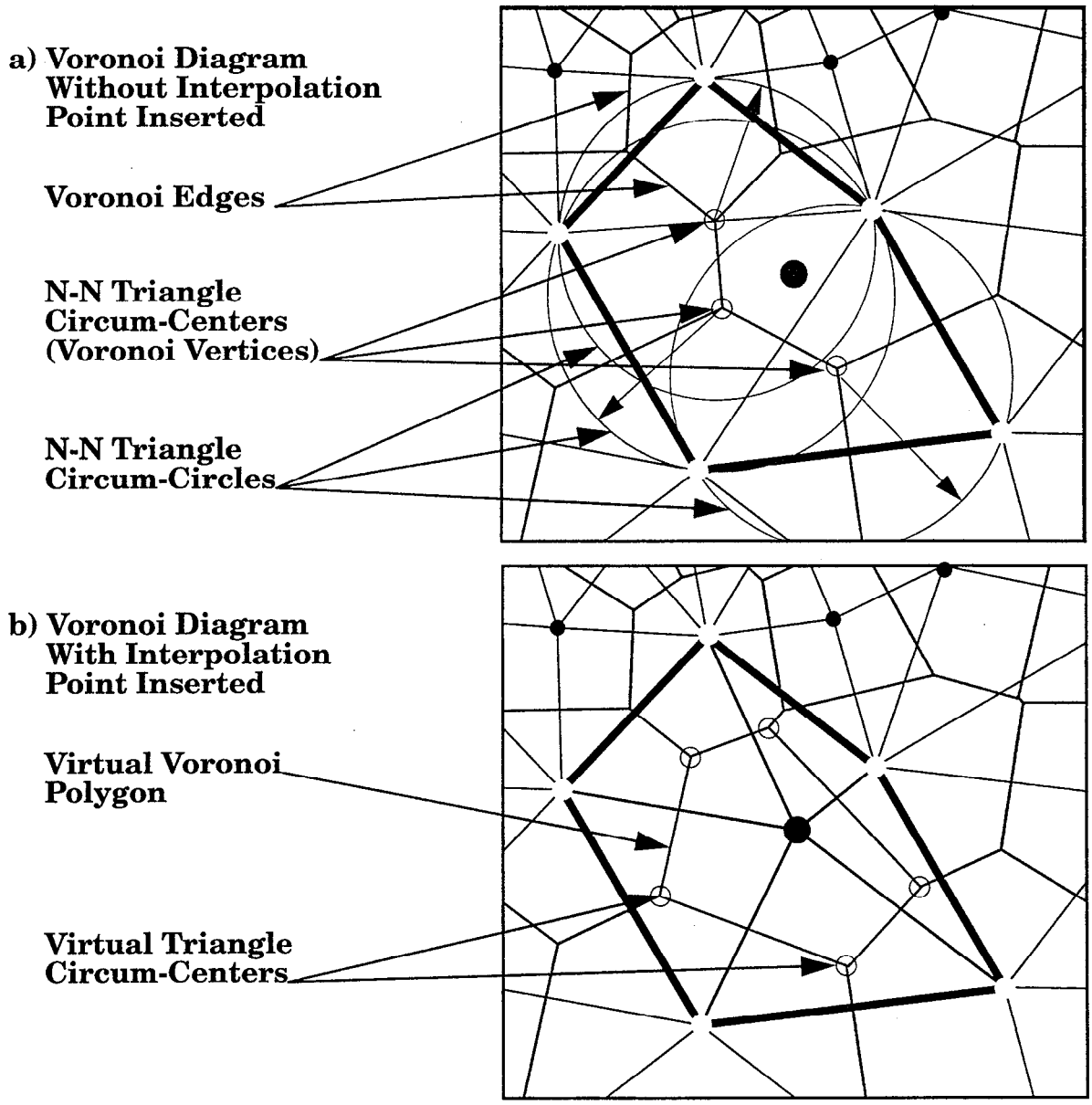


Figure D-4. Voronoi Diagrams With and Without Interpolation Point.
a) Shows the Voronoi diagram for the example tessellation (blue lines). Each polygon represents a single node from the tessellation. The vertices are the circum-centers of the circum-circles that inscribe each triangle of the tessellation. The N-N triangle circum-circles are shown in this panel along with their respective circum-centers. b) Shows the result of adding the interpolation point as a "fictitious" node to the tessellation. The resulting virtual triangles and voronoi polygon is depicted. The vertices of the virtual Voronoi polygon (red circles) are the circum-centers of the virtual triangles created when the interpolation point is "fictitiously" inserted into the mesh. The virtual circum-circles are not drawn to enable a clear view of the virtual Voronoi polygon.

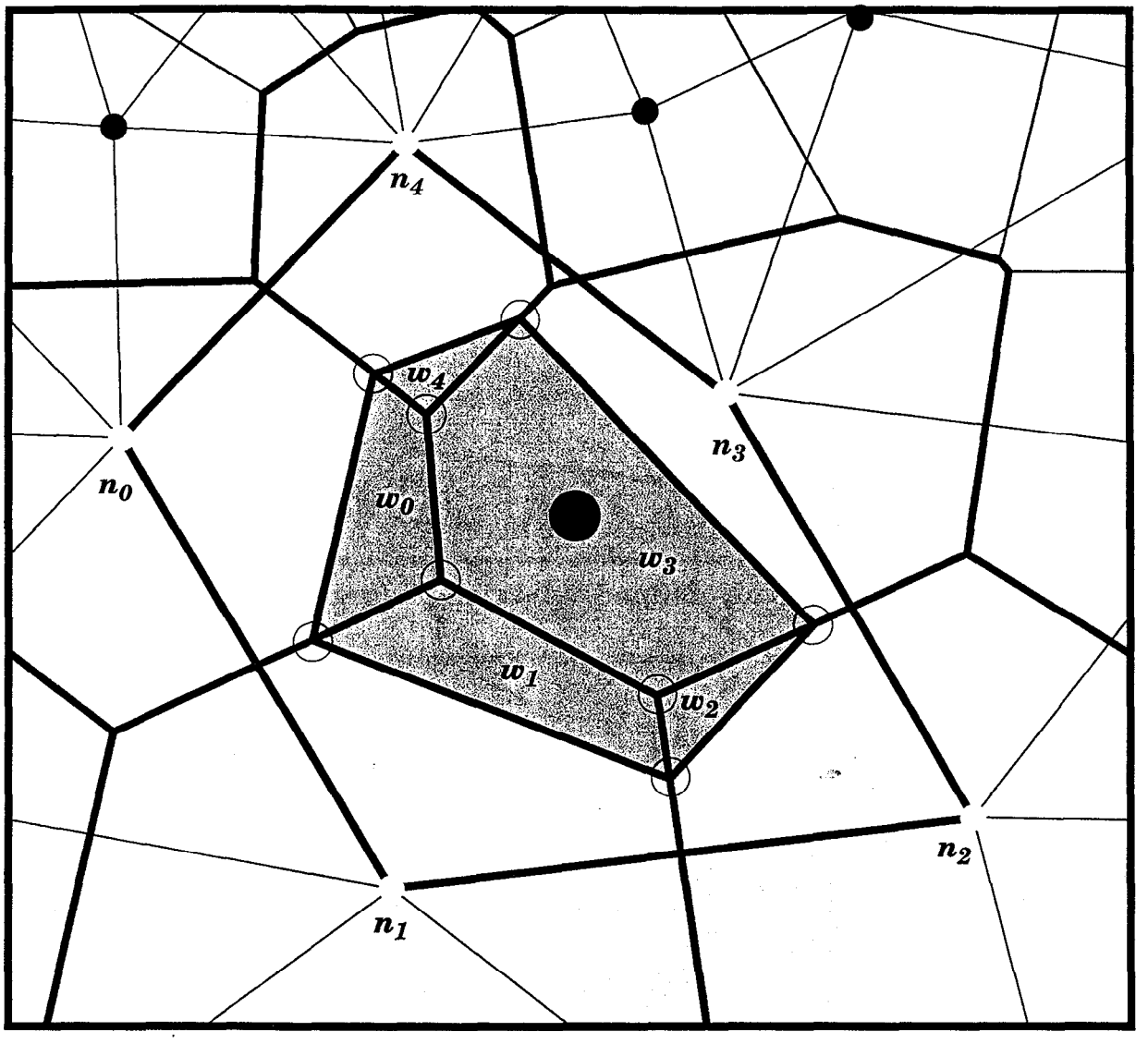
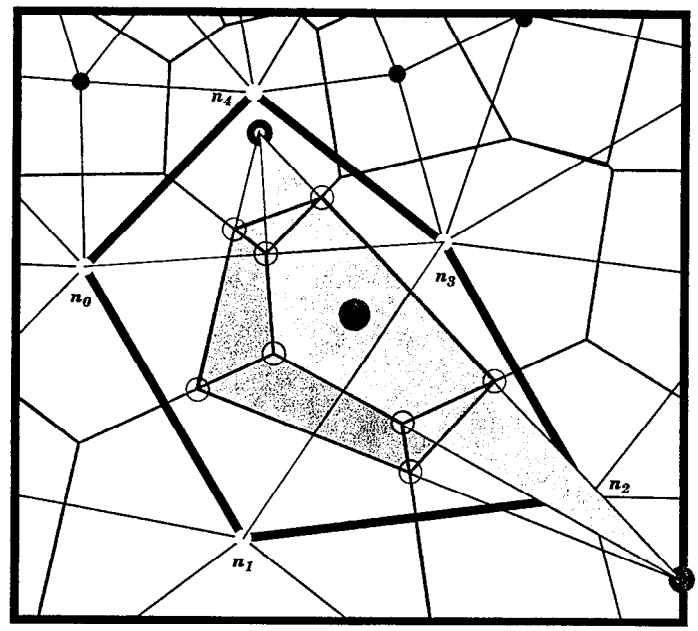
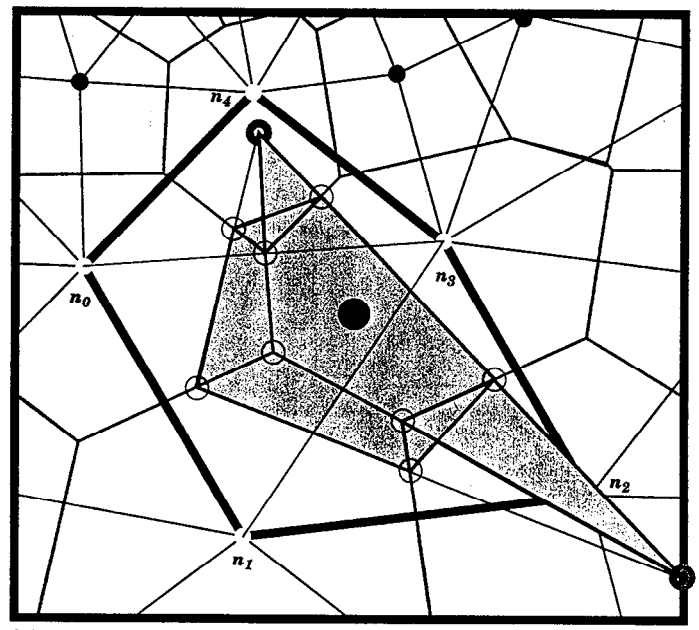


Figure D-5. Nodal Weight Definition: Overlapping Voronoi Polygons.
The respective weights of each of the N-N nodes can be determined from a normalized measure of the relative areas of overlap between the virtual voronoi polygon (red) and each of the N-N nodes voronoi polygons (blue). Each weight, w_i, is shown with its matching node, n_i, above.



a) positively defined sub-triangle (green)



b) negatively defined sub-triangles (grey)

Figure D-6. Sub-Triangle Weight Calculation Method
a) The green triangle represents a positively signed sub-triangle developed from the weight calculation prescription outlined by Watson (1992) and Sambridge (1995). b) shows two negatively signed triangles (grey). The sum of the three triangles produces the weight for node n_3. The method is indeterminate, however, if the interpolation point should fall on one of the N-N triangles' shared edges (the green edges). In that case the circum-circle formed from the two N-N nodes and the interpolation point is undefined since the circum-center (the green circles) resides at infinity.

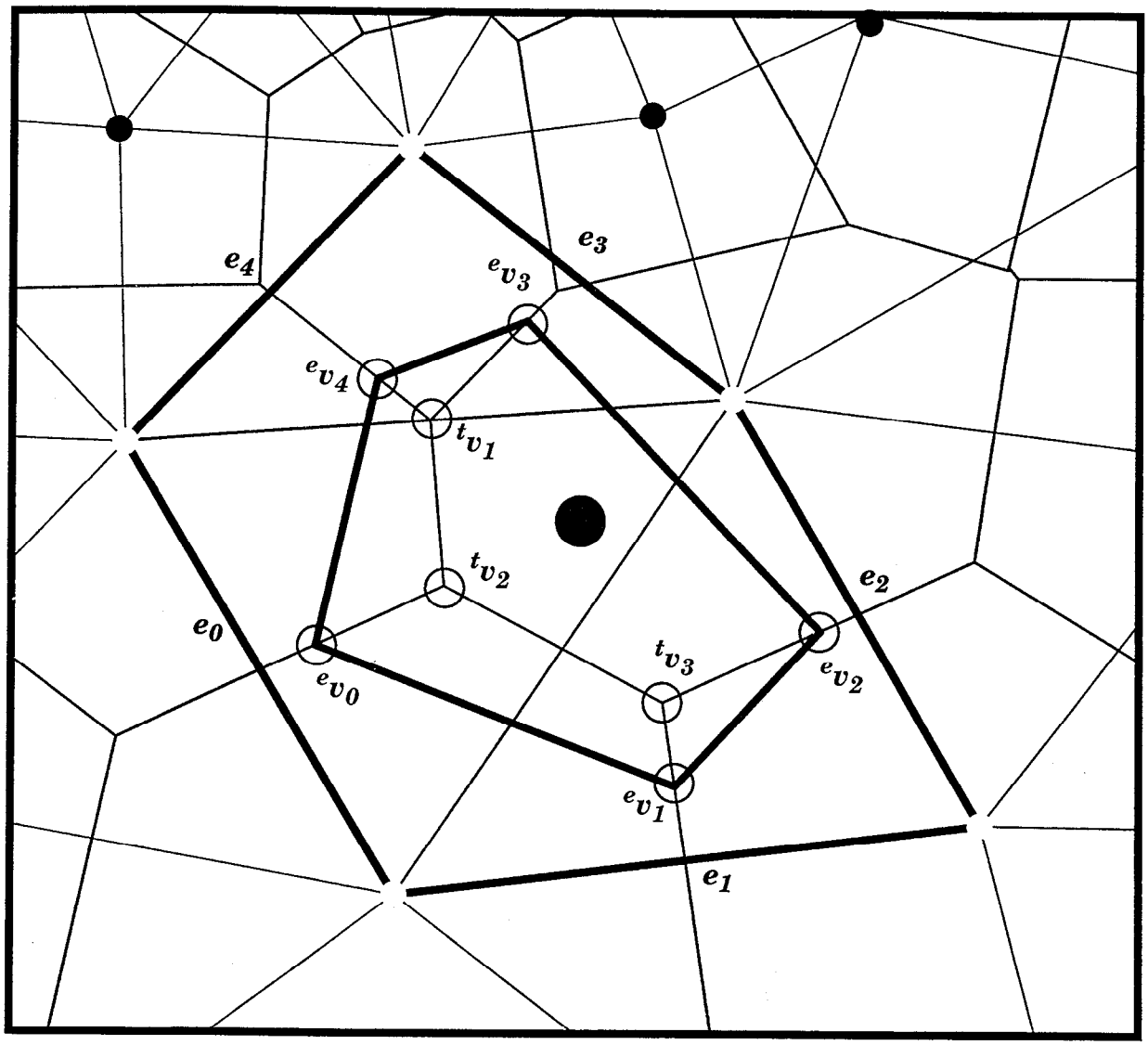


Figure D-7. Boundary Edge Matching With Virtual Voronoi Vertices
Each boundary edge can be matched to a specific virtual voronoi polygon vertex. The specific matching for a boundary edge is defined as the vertex that was created from the circum-circle of the virtual triangle that was constructed between the specified edge and the interpolation point. Each boundary edge, e_i, owns exactly one virtual vertex, $^{e}v_i$. Additionally, each N-N triangle has exactly one circum-center defined by $^{t}v_j$. The virtual polygon voronoi vertices (red) and the N-N triangle voronoi vertices (blue) completely define the geometry of the overlapping voronoi regions.

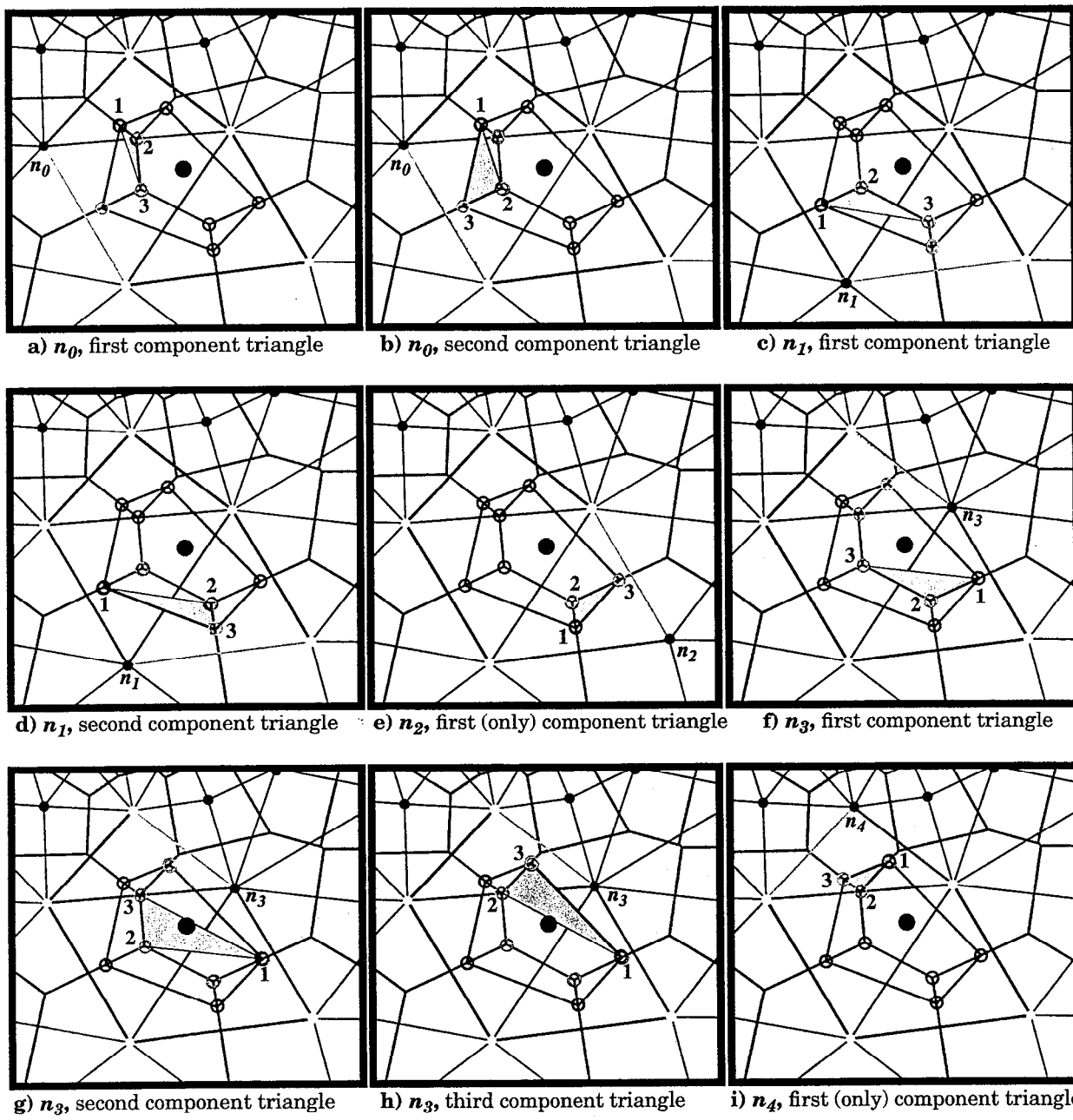


Figure D-8. Natural-Neighbor Weight Determination
The panel sequences illustrate the method of evaluating nodal weights by sequentially following N-N nodes around the enclosing polygon in a ccw fashion. The weights for each node are the sum of the individual component triangles that form overlapping areas between the interpolation points virtual voronoi polygon and the N-N node's voronoi polygons.

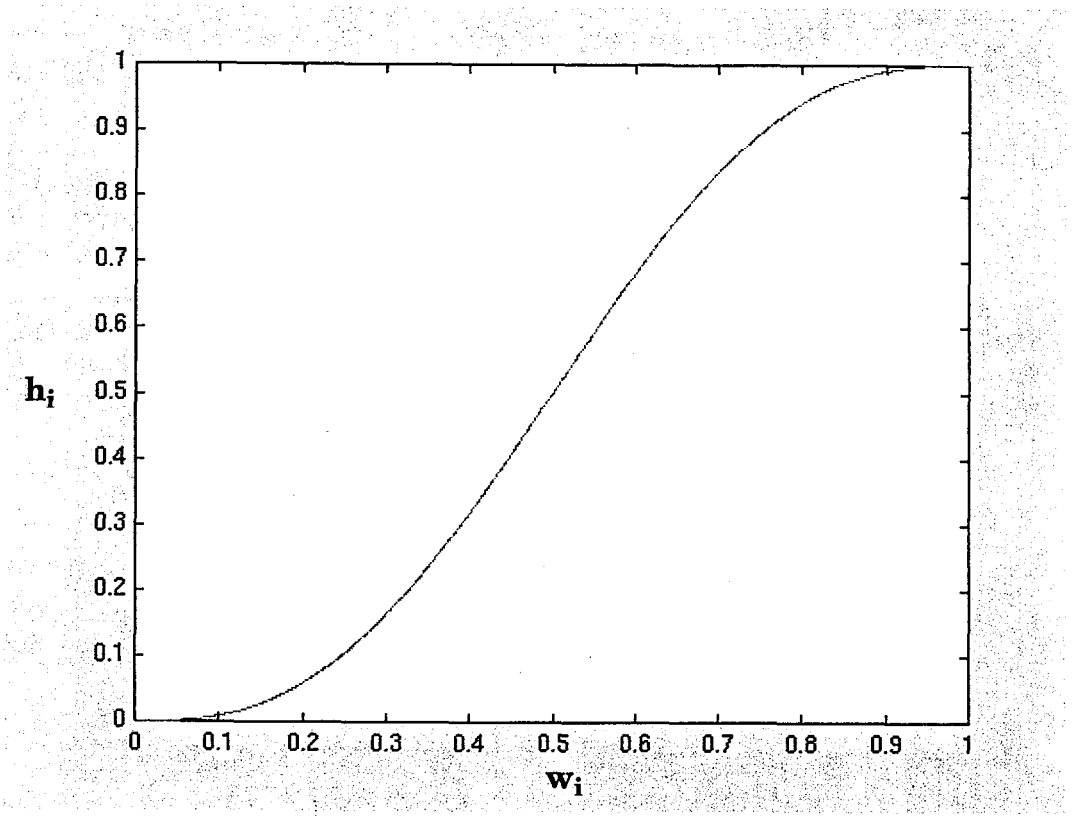
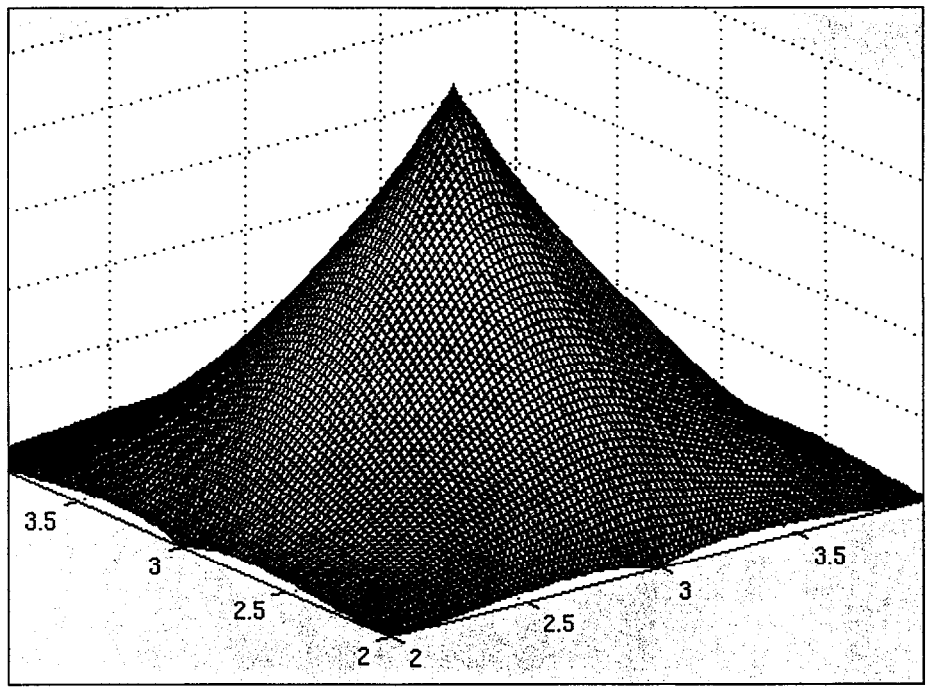
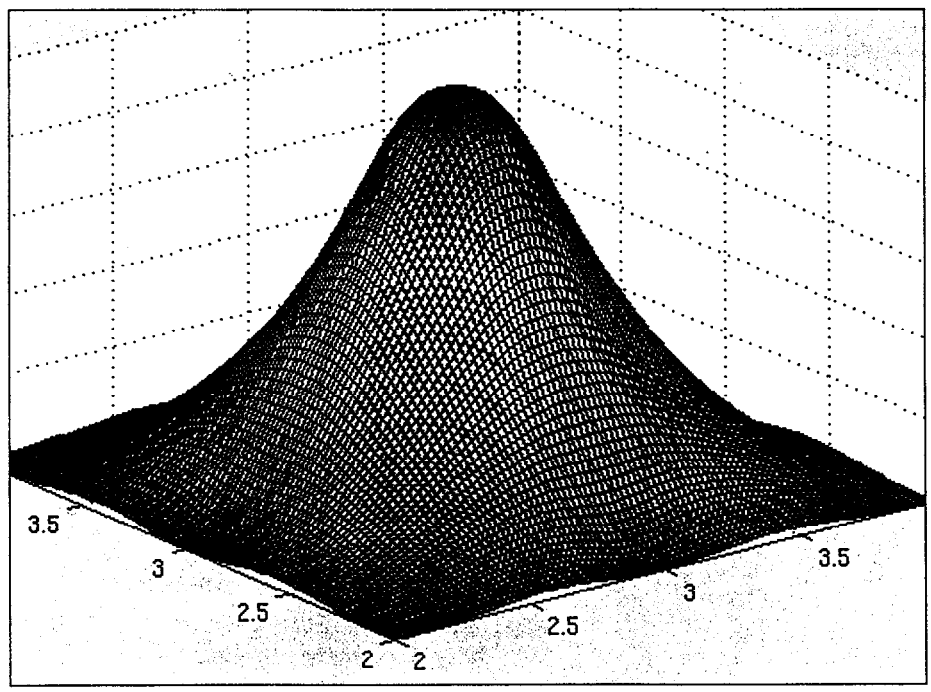


Figure D-9. Typical Modified Gradient Blending Function.
Typical GM NNI blending functions vary from 0 to 1 over the node-weight range of 0 to 1. The functions are always monotonically increasing and always have zero slope at $w_i = 0$. The slope at wi = 1 can vary from 0 to 1 or greater and can, if desired, be controlled by the nodes surface roughness coefficient. Generally, this is accomplished by setting the slope to be proportional to the surface roughness.



a)



b)

Figure D-10. Natural-Neighbor Basis Functions w/wo Gradient Modification

a) Linear Natural-Neighbor Interpolation (NNI) basis function illustrating non-differentiability condition at a node point. b) Same NNI surface changed with the addition of the Gradient Modification.

Appendix E: Matlab Packages

All of the data analysis, kriging, mesh refinement, and binary data file creation functions are written in Matlab. Though some of the functions could realize significant performance improvements if coded in a non-interpreted language (e.g. C or C++), we have found Matlab to be an excellent software development and research application environment. The huge library of built in functions, integrated graphics, and interpreted interface allow us to quickly develop sophisticated prototype functions, which in most cases are not so numerically demanding as to impact performance noticeably. Some numerically intensive software components have performance ranges that are near the limit of what is acceptable. In the future our intent is to replace these algorithms with substitutes written in C or C++ which will be bound to the Matlab interface using the Matlab CMEX facilities.

The high level functions are listed below, grouped by purpose.

E-1 Variogram Analysis

Functions used for variogram analysis:

- **vario_snl.m** - creates plots of the variogram and number of pairs per bin for a data matrix.

E-2 Kriging

Functions used for Modified Bayesian Kriging. Numerous sub-functions are also included:

- **make_blend.m** - makes a blending function structure (can be used for both blending functions and correlation coefficient functions).
- **draw_blend.m** - draw general shifted blending function (can be used for both blending functions and correlation coefficient functions).
- **krig_random.m** - Nearest-neighbor Modified Bayesian Kriging for values and errors on an arbitrary set of grid points (Grid) with boundary segment influence. This function factors in the influence of measurement error (smoothing) and uses nearest neighbor influence for performance enhancement.

- **krig_uniform.m** - Nearest-neighbor Modified Bayesian Kriging for values and errors on a regular grid (Glat, Glon) with boundary segment influence. This function factors in the influence of measurement error (smoothing) and uses nearest neighbor influence for performance enhancement.

E-3 Mesh Refinement

Functions used to create the and analyze the optimal tessellation for NNI given a set of Kriging parameters. Again, many additional subfunctions are included but not listed here:

- **refine_mesh.m** - refine_mesh calculates a refined mesh given 1) an initial grid description with data and accuracy refinement criteria, or 2) a predefined surface with a valid boundary description. In the second case a new surface can be fit into an existing grid which meets accuracy requirements for some set of previously defined kriged surfaces.
- **draw_krig_surf.m** - this function draws the kriged surface and an overlay of the tessellation given in tess_defn on top of the kriged surface.
- **check_surf.m** - this function is used to compare the actual kriged surface with it's approximate interpolated surface. The output contains the difference between the two for all points defined in the lat, lon pairs.
- **kbi_roff.m** - Function kbi_roff constructs the binary Read-Once Flat-File (ROFF) used by the C++ KBI code.

Appendix F: Locator Parameters for KBI

In order to use Knowledge Base data, additional parameters or specific settings for existing parameters are needed for the programs that use the libloc package (loc-SAT and Evloc). Here are the parameters with explanations:

use_kbi (default FALSE)
#Use Knowledge Base Interpolation - turn on to access the Kbase; if this is not on, the other KBI parameters are irrelevant

dist_var_wgt (default FALSE)
#Distance variance weighting - must be turned on to use any kind of a priori error information, which includes libKBI

num_dof (default 0)
#Number of degrees of freedom - must be set high (e.g. 9999) to get a coverage ellipse, i.e. an ellipse that reflects the a priori error information,

kb_shape_radius
#Knowledge Base shape radius - radius of circle (in degrees) for which to fetch information from the Knowledge Base

kb_dbtype (default 2)
#Knowledge Base database type - 1 = SDE; 2 = ROFF;

kb_bmodel
#Knowledge Base 'base model' - for correction data

kb_server
#Knowledge Base database server - needed if kb_dbtype = 1 (SDE)

kb_instance
#Knowledge Base database instance name - needed if kb_dbtype = 1 (SDE)

kb_db
#Knowledge Base database name - needed if kb_dbtype = 1 (SDE)

kb_usr
#Knowledge Base database username - needed if kb_dbtype = 1 (SDE)

kb_passwd
#Knowledge Base database password - needed if kb_dbtype = 1 (SDE)

kb_ff_name
#Knowledge Base flatfile name - path and name of Read Only Flatfile (ROFF), if kb_dbtype = 2

(This page intentionally left blank)